

平成28年 卒業研究論文

疑似ラインセンサーを用いた
S P レコード音検出

北海学園大学工学部 電子情報工学科
魚住研究室

4513213
クーン・トビアス

2016年10月11日

目 次

第 1 章 はじめに	3
第 2 章 概要	4
2.1 実験装置	4
2.1.1 接続図	4
2.1.2 CCD カメラ	5
2.1.3 パルスステージ	5
2.2 音溝形状と光照射	6
2.3 プログラム構造	6
第 3 章 部分読み出し法	9
3.1 撮像段階での問題	9
3.1.1 重複部分問題	9
3.1.2 湾曲問題	12
3.1.3 うねり問題	14
3.2 部分読み出し法の利点・欠点	15
第 4 章 画像データ生成	16
4.1 SPRecAnalyzer	16
4.1.1 物理的距離検出	16
4.1.2 自動撮像機能	16
第 5 章 画像処理	17
5.1 SpliceBot	17
5.1.1 処理内容	18
5.2 SigExBot	19
5.2.1 処理内容	19
5.2.2 信号修正機能	21
5.2.3 GapFilling	21
5.2.4 TryToRecover	24
5.3 Needle	24
5.4 fftf.m	24
第 6 章 結論	25

付 錄 A 物理的距離検出プログラム	26
A.1 PixelProcessing.m	26
A.2 ObjectExtraction.m	27
付 錄 B 画像処理プログラム	30
B.1 SpliceBot.m	30
B.2 SigExBot.m	34
B.3 Needle.m	44

第1章 はじめに

ここをメモ用に使おう。この論文の構造について考えたいと思い。

-- 概要 --

- 実験装置
- 音溝形状と光照射
- プログラム構造

-- 部分読み出し法--

- 撮像段階での問題
- 部分読み出し法の利点・利点

-- 画像処理--

- SpliceBot
- SigExBot
- Needle

-- 結論 --

-- 結論 --

- 少なくとも一つの箇所を完璧に書こう。今日は部分読み出し法のところに挑もう。

第2章 概要

本研究で用いた実験装置、撮像法とプログラム構造について説明する。

2.1 実験装置

SP レコードの撮像に用いられる実験装置を図 2.1 に示す。レコードの下にあるパルスステージはレコードをカメラの位置に対して平行に（図 2.1 で言うと左右に）動かす。この平行移動をパルスモータ F が実装する回転運動と組み合わせると、SP レコードの全領域を顕微鏡の下に持ってこられる仕組みができる。

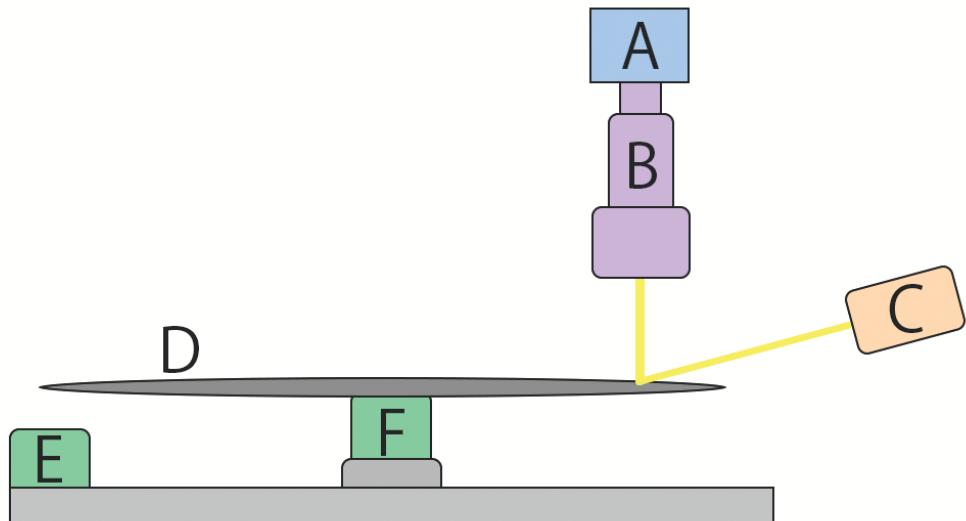


図 2.1: 実験装置の設定

2.1.1 接続図

それぞれの実験装置がどのようにしてパソコンと繋がっているかを図 2.2 に示す。GigE バスはパソコンの NIC (Network Interface Card) と接続される。

- A CDD カメラ
- B 顕微鏡
- C 照射装置
- D SP レコード
- E パルスモータ、平行移動用
- F パルスモータ、回転用

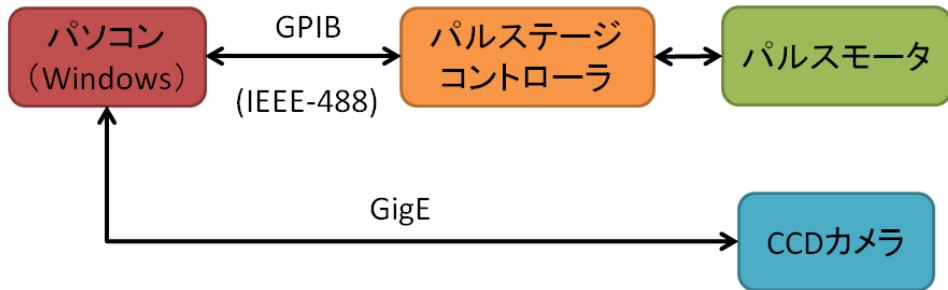


図 2.2: 実験装置の設定

2.1.2 CCD カメラ

本研究で使用されているカメラは JAI 社の「BB-500GE」である。カメラの最大解像度は 2456(h)x2058(v) であるが、Variable Partial Scan 機能を利用して実際にデータとしてパソコンに送られるのはもっと幅が狭い領域のデータである。表 2.1 にカメラのパラメータの設定を示す。

画像データ生成段階で SP レコードを一定の速度で回転させながら順次に撮像を行うため、ExposureTimeRaw パラメータ（無単位）は実験的にぶれが生じないよう設定した。表 2.1 のパラメータは全て自作の C# プログラムで設定される。SoftwareTrigger を実行して画像一枚取得するためのコードを図 2.3 に示す。

```
// We need to "pulse" the Software Trigger feature in order to trigger the camera!
myCamera.GetNode("SoftwareTrigger0").Value = 0;
myCamera.GetNode("SoftwareTrigger0").Value = 1;
myCamera.GetNode("SoftwareTrigger0").Value = 0;
```

図 2.3: C# プログラムの中からカメラの SoftwareTrigger を実行させるためのコード

2.1.3 パルスステージ

パルスステージの仕様を図 2.2 に示す。

表 2.1: JAI BB-500GE のパラメータの設定

ROI 範囲設定	2456 (h) x 32 (v)
GainRaw	550
ExposureTimeRaw	222
AcquisitionMode	Continuous
ExposureMode	EdgePreSelect
PartialScan	Variable Partial Scan
VariablePartialScanStartLine	1013
VariablePartialScanNumOfLines	32
TriggerSelector	FrameStart
TriggerMode	On
TriggerSource	Software
LineSelector	CameraTrigger0
LineSource	SoftwareTrigger0
LineInverter	ActiveHight

表 2.2: パルスステージの仕様

degrees per pulse (回転運動)	0.0025
mm per pulse (平行移動)	0.002

2.2 音溝形状と光照射

顕微鏡を通って撮像される光から SP レコードの音溝に記録されたデータを復元するのが本研究目標であるから、撮像されるデータに音溝の形状が何らかの方法で読み取れなければならない。これを成し遂げるために、光を斜めに照射して、音溝の壁の一部が明るく映るようにするのが有力な方法であることが過去の研究でわかつた (ref!)。図 2.4 に光の反射具合を模式的に表す。

図 2.4 を見てわかるように、SP 盤に対して垂直に反射する光は、音溝の壁部分が一番多い。一方、音溝以外の平面部分で反射する光は垂直に反射せず、顕微鏡にはほとんど入らない。この設定でカメラと顕微鏡で撮像した画像を図 2.5 に示す。

2.3 プログラム構造

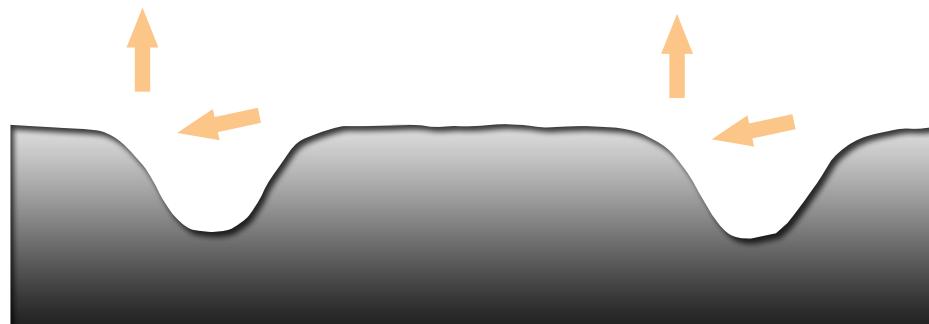


図 2.4: 反射の仕方

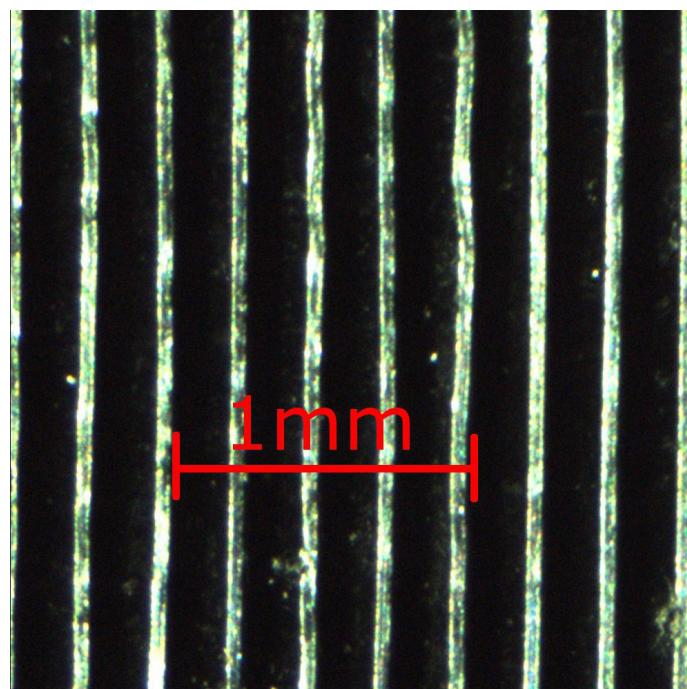


図 2.5: SprecMicroscopeImgLabeled

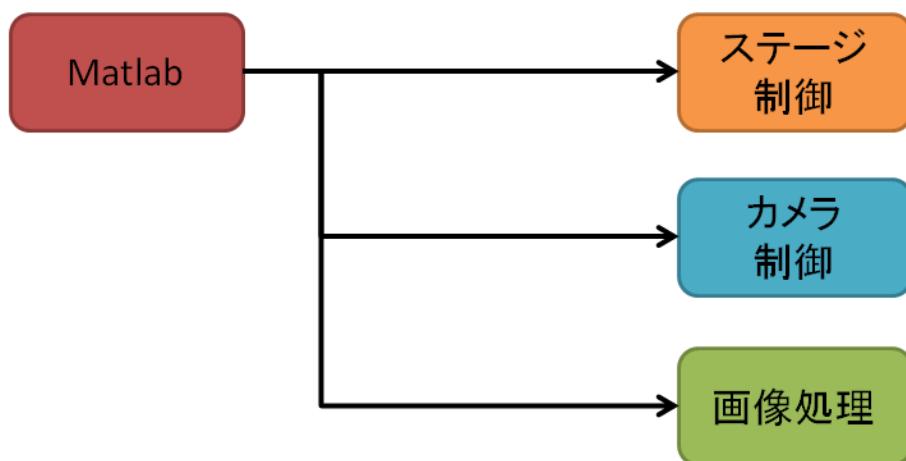


図 2.6: Prog Setup Old

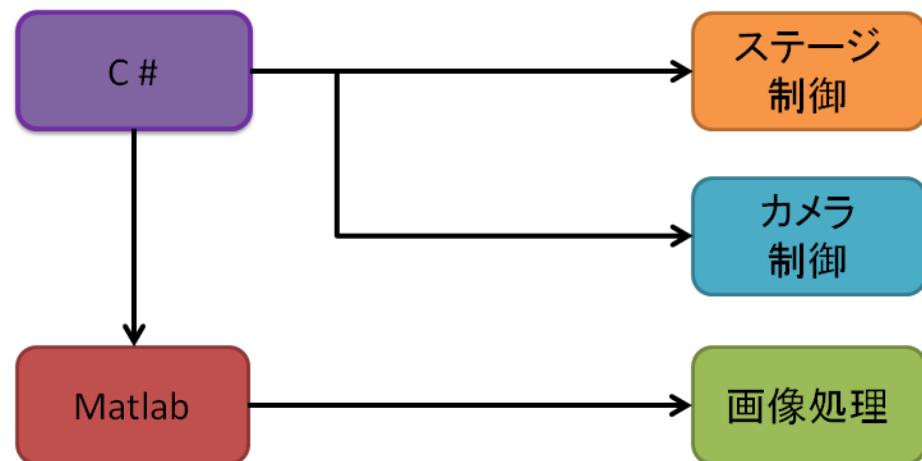


図 2.7: Prog Setup New

第3章 部分読み出し法

カメラが撮影した画像データの一部しか用いらない部分読み出し法には利点もあれば欠点もあり、撮像段階で直面する問題とその解決の説明を通じてそれらについて考察する。

3.1 撮像段階での問題

画像処理技術を適応し音溝から音データを抽出するためには、画像上の音溝データが連続的で、余分な重複などがないものでなければならぬ。四角形の画像を撮影するカメラでその下を回る円盤の表面を撮像する際どのような問題が起きるかを調べて、解析する。

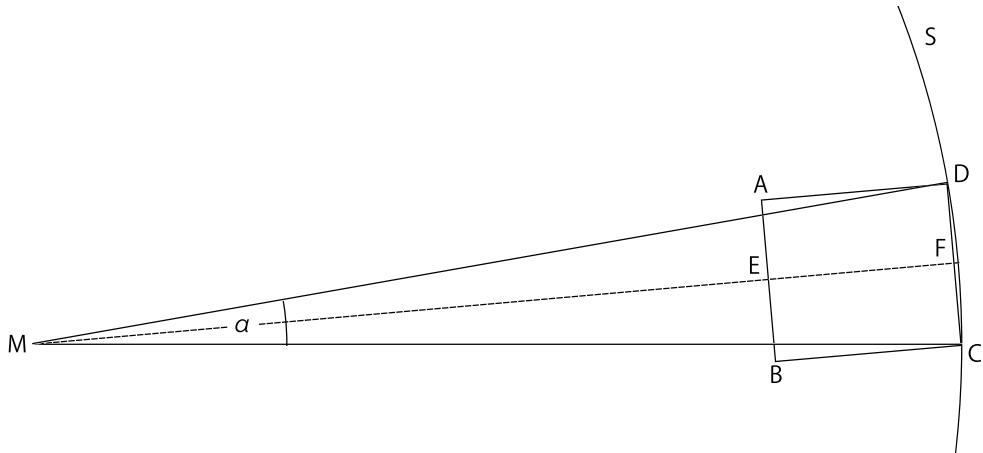


図 3.1: 撮像の様子を幾何学的に表した図

図 3.1において、四角形 ABCD がカメラの撮像範囲、S が SP レコードの外側、M が SP レコードの中心、 α が次の画像を取るための回転角度だとする。また $\overline{EG} = \overline{GF}$ である。

3.1.1 重複部分問題

図 3.2 の四角形 A, B, C が順番に撮像された画像範囲だとする。このとき、それぞれの撮像範囲が重なることがわかる。撮像範囲が四角形であることを前提とする

と、画像をどのように撮像しても重複部分、または情報損失が生じる。音溝から音を抽出する画像処理段階で、情報損失があってはならない。重複部分も画像処理を困難にするため何らかの方法でそれを最低限に抑える必要がある。

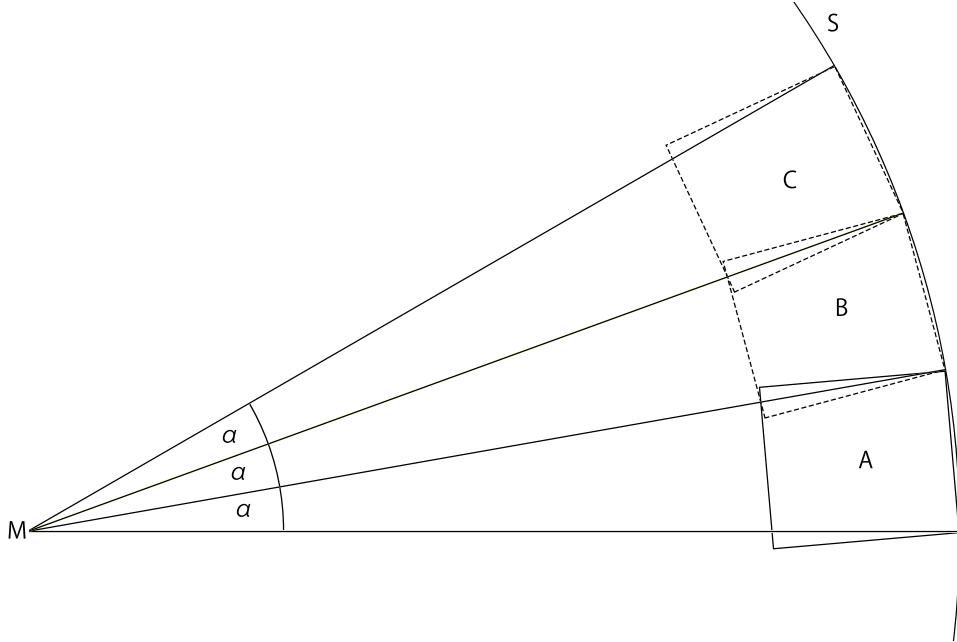


図 3.2: A, B, C と順番に撮像を行った際に重複部分が生じる

本研究では、撮像範囲の幅（図 3.3 中 $\overline{AB} = \overline{CD}$ ）を小さくすることで重複部分の影響を無視できる程度に抑える。どのくらいの幅であれば無視できる程度になるかを調べることにしよう。図 3.3 中の ϵ が一画像の片方の重複部分だとする。隣り合っている画像でのピクセルの重複部分を考えると 2ϵ となる。

ϵ を求めるために、まず直角三角形 MCF より $\frac{\alpha}{2}$ を求めておく。

$$\frac{\alpha}{2} = \arctan \left(\frac{\overline{BE}}{\overline{MF}} \right) \quad (3.1)$$

$\frac{\alpha}{2}$ を用いて以下の方程式が成り立つ。

$$\overline{BE} - \overline{BG} = \tan \left(\frac{\alpha}{2} \right) \overline{ME} \quad (3.2)$$

$$\epsilon = \overline{BG} = \overline{BE} - \tan \left(\frac{\alpha}{2} \right) \overline{ME} \quad (3.3)$$

式 3.1 を式 3.3 に代入し式を整理すると

$$\epsilon = \overline{BE} \left(1 - \frac{\overline{ME}}{\overline{MF}} \right) \quad (3.4)$$

が得られる。本研究で作成したC#プログラムの中のキャリブレーション機能を利用した後のパルスステージだと、変数 \overline{BE} , \overline{ME} と \overline{MF} は常に把握できていることから、SPレコードの任意の位置においての ϵ を求めることができる。 ϵ の性質を調べるために、まずカメラの最大解像度で撮像を行った際の計算を進めてみる。ただし、撮像場所においてはSPレコードの一番外側と内側と2つのケースに分けることとする。カメラに直結している顕微鏡の拡大率が1.5xに設定してあるときの次の定数を計算に用いる。

$$\gamma = 0.002366 \left[\frac{\text{mm}}{\text{pixel}} \right] \quad (3.5)$$

$$\delta = 422.65 \left[\frac{\text{pixel}}{\text{mm}} \right] \quad (3.6)$$

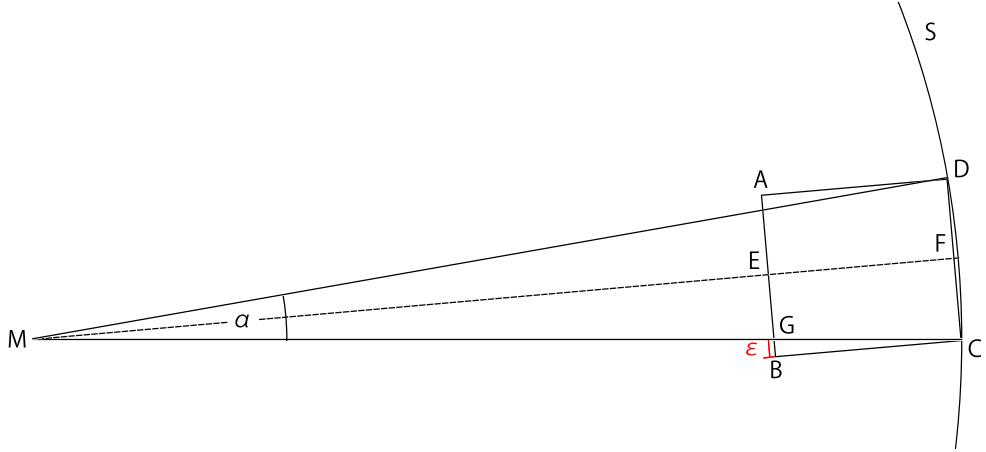


図 3.3: 重複問題を数値化する ρ の幾何学的図

最大解像度で撮像した画像は実際のサイズは

$$\text{height} = \overline{EF} = 5.81 \text{ [mm]} \quad (3.7)$$

$$\text{width} = 2 \cdot \overline{BE} = 4.87 \text{ [mm]} \quad (3.8)$$

となり、またレコードの外側で撮像したときの \overline{ME} は

$$\overline{ME} = 120.0 \text{ [mm]} \quad (3.9)$$

となる。 $\overline{MF} = \overline{ME} + \overline{EF}$ を考慮しながら式3.7, 3.8, 3.9を式3.4に代入して計算すると

$$\epsilon_1 = 0.1124 \text{ [mm]} \quad (3.10)$$

となり、式3.6の単位換算定数 δ をかけると

$$\epsilon_1 = 47.52 \text{ [pixel]} \quad (3.11)$$

という結果が得られる。同様にSPレコードの内側での ρ を $\overline{ME} = 50.0 \text{ [mm]}$ として求めると

$$\epsilon_2 = 0.2534 \text{ [mm]} = 107.14 \text{ [pixel]} \quad (3.12)$$

となる。ここで、前述したように、隣り合っている画像の隣接重複距離が 2ϵ であることを思い出すと最悪の場合の隣接重複距離が

$$2\epsilon_2 = 0.5068 \text{ [mm]} = 214.28 \text{ [pixel]} \quad (3.13)$$

となり、これだけのピクセル数の情報が重複していると無視できない問題と判断せざるを得ない。しかしながら、上記の計算は画像の縦幅 $\overline{AB} = \overline{DC}$ がカメラの最大解像度設定にしてある際の ϵ_3 の計算結果。縦幅32ピクセルに指定した際の計算は以下の通りになる。

$$\text{width} = 2 \cdot \overline{BE} = 0.0378 \text{ [mm]} \quad (3.14)$$

$$2\epsilon_3 = 0.00782 \text{ [mm]} = 3.331 \text{ [pixel]} \quad (3.15)$$

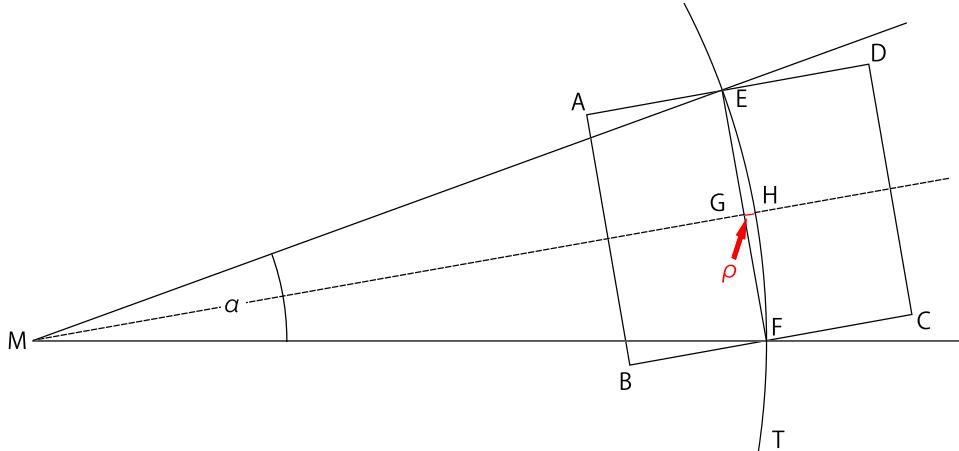
式3.15の結果は最悪条件の下で計算したものであることを考慮しながら、実際の撮像データで幅3ピクセルの音溝に大きい変化があり得ないことが確認でき、このくらいの重複であれば無視してもいいと判断できる。

3.1.2 湾曲問題

SPレコードに刻まれた音溝がらせん状にレコードの中心へと近づいていく仕組みであるからレコードの任意の位置で音溝の形状を撮像した際にそれが湾曲していることが画像情報に残される。複数の連続的に撮像された画像を並べてみると音溝の湾曲成分が周期的な上下振動として残り、音検出段階でこのゆっくりとした振動が間違って音情報として検出されることを防ぐために、湾曲の影響を最低限に抑える必要がある。

図3.4において、四角形ABCDがカメラの撮像範囲、TがSPレコードの音溝、MがSPレコードの中心、 α が次の画像を取るために回転角度だとする。また、 $\overline{FG} = \overline{GE}$ 、角MGFは直角である。

$$\overline{GF} = \frac{1}{2}\overline{AB} \quad (3.16)$$

図 3.4: 音溝湾曲を数値化した変数 ρ

$$\frac{\alpha}{2} = \arctan \left(\frac{\overline{GF}}{\overline{MF}} \right) \quad (3.17)$$

$$\overline{MG} = \cos \left(\frac{\alpha}{2} \right) \overline{MF} \quad (3.18)$$

$$\rho = \overline{MH} - \overline{MG} \quad (3.19)$$

式 3.16～式 3.19 より

$$\rho = \overline{MH} \left\{ 1 - \cos \left[\arctan \left(\frac{\overline{AB}}{2\overline{MH}} \right) \right] \right\} \quad (3.20)$$

重複問題の計算と同様にレコードの一番外側と内側で ρ を求めることにする。まず、画像の横幅を 32 ピクセルと設定し、実際の画像幅を

$$\text{width} = \overline{AB} = 0.0378 \text{ [mm]} \quad (3.21)$$

と置いておく（単位換算には式 3.6 の γ を用いる）。レコードの外側だと

$$\overline{MH} = 120.0 \text{ [mm]} \quad (3.22)$$

になる。このときの湾曲問題係数 ρ は

$$\rho_1 = 0.001488 \text{ [\mu m]} = 0.000629 \text{ [pixel]} \quad (3.23)$$

レコードの内側では

$$\overline{MH} = 50.0 \text{ [mm]} \quad (3.24)$$

なので、湾曲問題係数は

$$\rho_2 = 0.03572 [\mu\text{m}] = 0.0015 [\text{pixel}] \quad (3.25)$$

となる。比較のため、以下に悪条件（レコードの内側での撮像）のもとで画像幅 2058 ピクセル ($\overline{AB} = 4.87 [\text{mm}]$) のときの湾曲問題係数を示す。

$$\rho_3 = 59.18 [\mu\text{m}] = 25.0 [\text{pixel}] \quad (3.26)$$

以上の計算より、横幅 32 ピクセルで画像撮像を行った場合の湾曲問題による影響が極めて小さく、無視できる程度であることがわかる。

3.1.3 うねり問題

うねり問題はレコードに開けられた穴が偏心しているために起きる。ここまで幾何学的问题はレコードの性質（音溝がらせん状に円形のレコードに刻まれている）やカメラの撮像範囲が四角形に対し、レコードがカメラの下で回転することに起因している。ただし、うねり問題は湾曲問題と重複問題と異なり、不可避の問題ではない。完璧に作られたレコードをそれにぴったりと合う実験装置であれば、うねり問題は起きないと見える。

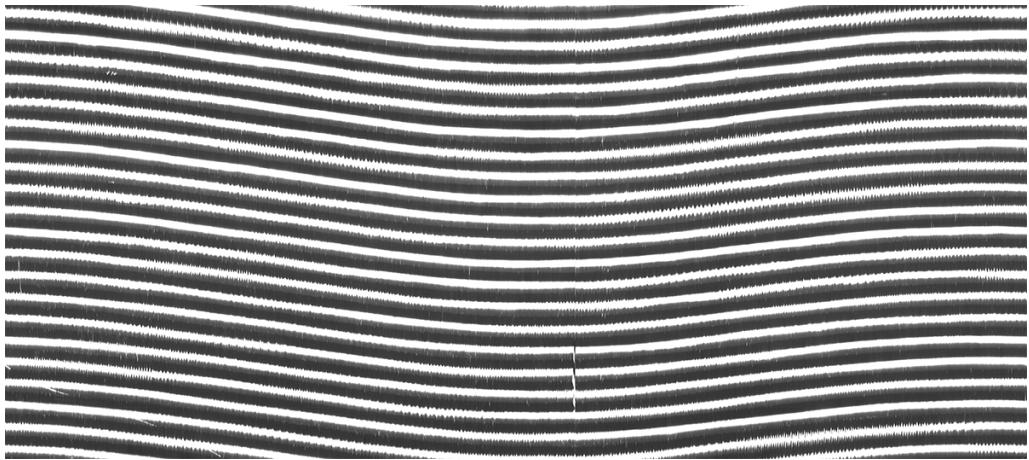


図 3.5: レコード一週分の画像データ。横幅は圧縮された。

図 3.5 にレコードの一週分の画像データを示す。ページ内に収まるように、横幅は圧縮されました。この画像データに見られる音溝の形を確認してみると、ゆっくりと上下振動していることが分かる。この振動は音情報ではなく、単なる穴が偏心しているために起きる。本研究においては、うねり問題の影響を信号処理で抑制する。

3.2 部分読み出し法の利点・欠点

撮像時に直面するする問題とその解決法については前節に詳しく説明している。ただし、部分読み出し法の利点・欠点について議論を進められるようにそれらの問題をもう一度リストアップする。

- 重複部分問題
- 湾曲問題
- うねり問題

重複問題と湾曲問題は本研究と同じ実験設定ならば必ず台頭する。その解決法に関しては、画像処理を通じて克服する手法もあるが、部分読み出しを採用すれば重複問題と湾曲問題の影響が無視できる程度に収まる。これが部分読み出しの利点の一つである。

欠点として挙げられるのは画像結合が増えること。幅が非常に小さい（本研究では32ピクセル）画像データを数多く取得し、画像処理を通して連結させる過程において、隣り合っている画像データの境界面が重複も欠損もなく実際の音溝を連続的に再現できるかどうかが部分読み出し法の難点である。

第4章 画像データ生成

ここで SPRecAnalyzer について書く。映画を撮るか、画像を取るか。

4.1 SPRecAnalyzer

4.1.1 物理的距離検出

pixelprocessing.m 無料オープンソースツールボックス JSONLAB をインストールする必要がある

4.1.2 自動撮像機能

第5章 画像処理

SPRecAnalyzer を利用して取得した画像データは数多くの 2456x32 ピクセルの画像ファイルから構成される。原理的には Matlab で直接この画像データをメモリに読み込み画像処理の元にすることも可能ではあるが、画像処理アルゴリズムのデバグ作業などを考えると、この非常に幅が狭い画像データを一度もう少し幅が広い画像ファイルにまとめてから画像処理を行った方が画像データ生成メカニズムや後ほどの画像処理での結果確認が取りやすくなる。

幅が狭い画像データを数十枚束ねて幅が広い画像データを作成しておくのが SpliceBot アルゴリズムの仕事である。SpliceBot が作成した画像データを元に音溝に謎つて信号データを抽出するのは SigExBot。最後に SigExBot の信号データに信号処理を施して音信号の形に変形してくれるのは Needle。本研究の画像処理は全部 Matlab で行うことになっている。Matlab の有料ツールボックスのインストールは不要。

5.1 SpliceBot

Splice¹Bot は Matlab 用の画像合成アルゴリズム。SpliceBot が正常に作動するためには画像ファイル、フォルダ名とフォルダ構造を守らなければならない。アルゴリズムファイル「SpliceBot.m」と同じフォルダに「Round○○」フォルダを置く。○○は 1 から始まる画像データ生成の際の週番号。例えば、「Round1」フォルダにはレコードの一番外側を一周したときの画像データを格納する。アルゴリズムが要求する Round○○フォルダの中のフォルダ構造を図 5.1 に示す。



図 5.1: フォルダ Round○○ フォルダの中身

¹ 「splice」の意味合いについて。Oxford Dictionary より： Join (pieces of timber, film, or tape) at the ends

BWImages フォルダには 2 値化画像データが格納される。アルゴリズムの中では「ConvertToBWImages」メソッドがこのデータを作成する。次段階の画像処理アルゴリズムは 2 値化データを必要としないため、処理時間短縮を目指すならコメントアウトしてよいメソッドである。2 値化データはあくまでも画像データ確認の一つの手段にすぎない。

markedImages フォルダには SigExBot が追跡マークをプロットした後の画像データが入る。rawdata フォルダには画像データ生成で作成される画像データを格納する。画像データのファイル形式は「AIA ○○.bmp」。○○は 1 から始まる通し番号である。SpliceBot はこのフォルダの画像データを処理対象にする。splicedImages フォルダには SpliceBot が画像合成処理を行った後の画像データを格納する場所。画像合成後にできた画像ファイルを 5.2 に示す。

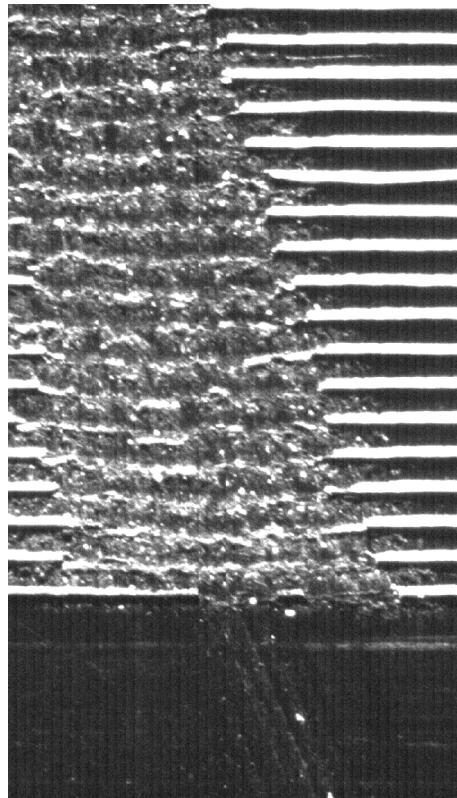


図 5.2: spliceBot が作成した合成画像データ（レコードの表面が削れた箇所）

5.1.1 処理内容

Matlab 環境内でパスを SpliceBot.m が格納されているフォルダに設定した後に Command Window に「spliceBot = SpliceBot();」を入力すると SpliceBot オブジェクトが生成される。生成時に、クラスファイルと同じフォルダにある Round ○○ フォルダが全てオブジェクト内に登録される。

アルゴリズムを実行するには「spliceBot.Start();」を入力する。実行開始後、全ての登録されている Round ○○ フォルダに対し、以下の処理が行われる。

1. フォルダ内の bmp ファイル数を検出
2. 画像データを 50 枚ずつ合成する
rawdata 画像データを束ねる前に rawdata 一枚ずつに対して以下の処理を行う：
 - (a) 幅 1 ピクセルの枠を削除する（画像データのサイズが縦横 2 ピクセル縮む）
 - (b) 上下反転、90 度回転、上下反転
3. 残り画像データが 50 枚以下の場合、残り画像数だけ 2. と同じ手法で合成する
4. splicedImages フォルダ内の.bmp ファイルの個数を検出
5. 合成画像データをグレースケールに変換する
6. 合成画像データを 2 値化する
7. 処理対象データパスを次の登録 Round フォルダに変更して 1. – 6. をフォルダがなくなるまで繰り返す

5.2 SigExBot

このアルゴリズムは本研究の画像処理の中で中心的な役割を果たしている。処理対象となるのは SpliceBot が作成した画像データ。SigExBot²は SpliceBot と同じフォルダ構造を要求する。本研究の画像処理アルゴリズムは殆ど例外なくみんなクラスファイルとして定義されている。従って、SigExBot も SpliceBot と同様にクラスのインスタンス化がまず最初に必要である。Matlab の Command Window に「sigExBot = SigExBot();」でクラスをインスタンス化し「sigExBot.Start();」で処理開始。

5.2.1 処理内容

「sigExBot = SigExBot();」とオブジェクトを作成したとき、以下の処理が行われる。

1. 様々のパラメータの初期化
2. Round ○○ フォルダの個数を検出し、処理対象フォルダを Round1 と設定する
3. Round1 フォルダ内の bmp ファイル数を検出
4. Round1 フォルダの検出された画像データを全部 RAM に読み込む

「sigExBot.Start();」と処理を開始させた後は以下の処理が実行される。ただし、信号修正機能については後ほど説明するため、ここでは除く。

² 「Signal Extraction Bot」の略称

1. 開始した直後は設定された画像データのピクセル値を左下から上へと向かって調べていき、音溝を探す
2. 縦軸に対する音溝の中心を計算し、現在の位置をそれに合わせる
3. 現在位置のピクセルに色を付ける（デバッグ用）
4. 設定されたステップサイズの値だけ右へと進む
5. 2. に戻り繰り返し処理を行う

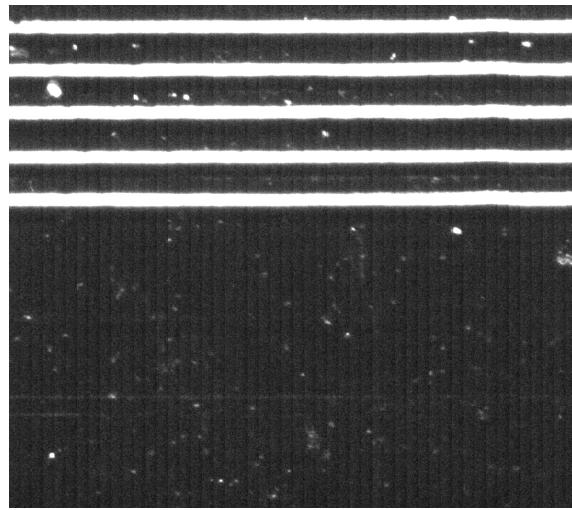


図 5.3: 処理開始直後アルゴリズムは左下から上に向かって音溝を探す

基本的な処理内容は上記のようになっている。ただし、右へと進んでいき画像データの左端に到達した際は次の画像が現在処理対象画像として設定される。また、画像データを何回か一周した後には画像の上部に到達し、ある一定のピクセル数を超えると Round 番号がインクリメントされ、次の画像データが RAM に格納され、上記の処理が繰り返し行われる。

なお、画像データ生成ソフトで Roundx と Roundx+1 の撮像縦座標の違いがちょうど 1500 ピクセルになるように設定されているため、Roundx から Roundx+1 へと進むときは、現在縦座標パラメータから 1500 ピクセルを引くことで、途切れることなく音溝追跡処理が続けられる。

4. のステップサイズに関して説明する。蓄音機でレコードを再生するとき、針が外側から内側へと向かっていくに連れて、針の直下を回るレコードの表面速度が減少する。同じ 1000Hz のサイン信号をレコードの外側と内側とで比較すると、内側の方が音溝の周期が短くなる。これは表面速度が減少しても、時間軸に対しては同じ 1000Hz の信号を保たなければならないからである。この音溝の横軸に対する収縮に対応するため、処理対象 Round フォルダが変わるとたびにステップサイズを調節する。中心に近づけば近づくほど、ステップサイズが小さくなる。

また、Round フォルダが変わるたびに、アルゴリズムの追跡マークが付けられた画像データが markedImages フォルダ内に保存される。これはあくまでもデバッグ用であり、字段階の信号処理アルゴリズムが必要とするデータではない。図 5.4 に追跡マーク付きの画像データを示す。したがって、処理時間短縮を目指すなら追跡マーク関連のメソッドをコメントアウトするとよい。

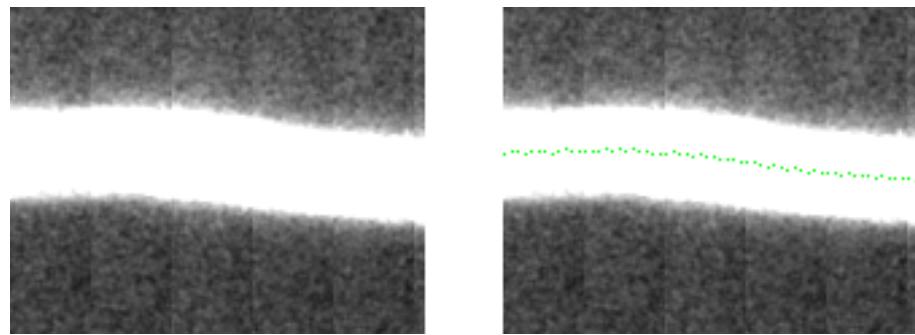


図 5.4: 元画像データ（左）とアルゴリズムが適応された後の画像データ（右）

図 5.4 に示された追跡マークは実際に検出されるデータと一致する。

5.2.2 信号修正機能

本研究で音検出対象となる SP 盤レコードには擦り傷、割れやひびが入っているものが多い。それに加え、音溝に反射される光のが音溝読み取り手段になるため、蓄音機と異なり単なるほこりにも影響を受ける。これらの悪影響に対応できるように音信号検出アルゴリズムに様々な工夫が施された。乱れた信号を修正するための機能を以下にリストアップする。

- GapFilling
- TryToRecover

Matlab プログラムとの関連性がわかるように機能名は英語で表記された。

5.2.3 GapFilling

GapFilling³機能はほこり、擦り傷やひびに対応するために実装された。この機能を説明するためにはまず最初に SigExBot の状態について記述する必要がある。SigExBot が取り得る状態を以下に示す。

- ライントレース中（緑）

³ギャック埋め立て

- 警戒中（赤）
- ギャップ埋め立て中（青）

かっこの中の色はデバッグ用の追跡マークの色に該当する。普段、音信号検出アルゴリズムは「ライントレース中」の状態にある。アルゴリズムが謎っていこうとしている白い線の幅が急激に⁴ピクセル以上変わる際に、アルゴリズム状態が「警戒中」へと変化する。警戒中の追跡点線は赤くなり、幅が平均的な値に戻らない限りアルゴリズムは白い線の中心を追わず幅の回復が確認できるまではまっすぐ進んでいく。なお、アルゴリズムが追いかける白い線は音溝の壁に反射する光にすぎず、当然のことながら、反射光で撮像する画像データ内の白い線の幅は一定にはならない。レコードが回りながら幅が伸縮する。自然と映り具合が多少変わる幅の変化と、ひこりやひびで幅が変わることを区別するためにアルゴリズムの中で平均的な幅を常に計算している。そのため、100ステップに渡って画像データ内の音溝幅が5ピクセル大きくなってしまってもアルゴリズム状態は「警戒中」とならず、追跡処理が普通通りに続けられる。

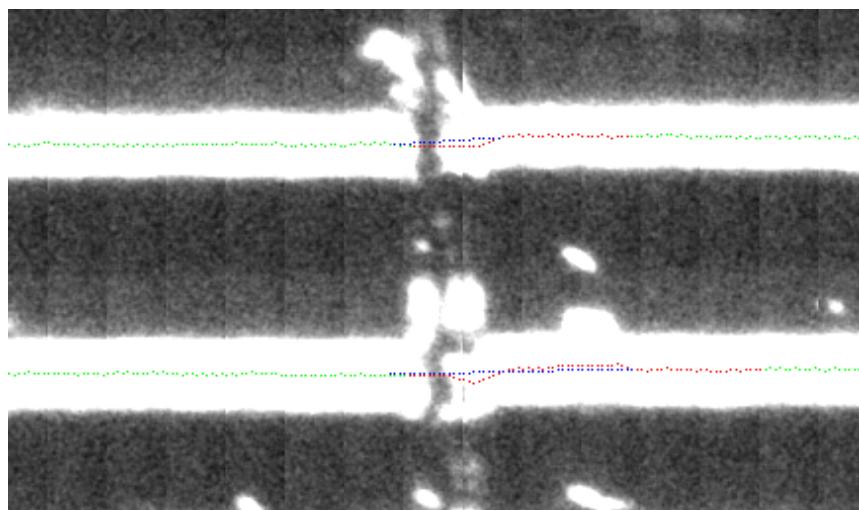


図 5.5: GapFilling 機能によるレコードに入っているひびで乱れたデータを修正

一旦アルゴリズム状態が「警戒中」となった後は音溝データの幅が平均値の5ピクセル以内に戻らなければ回復できない。音溝幅が平均値±5ピクセル以内になつたとしても、その状態が30ステップ続かなければならぬ。これらの条件が満たされれば、アルゴリズムは「警戒中」から「ライントレース中」に戻り、追跡マークが赤から緑に変わる。

しかし、警戒中で進んだ際の信号データが満足のできないものになる場合もあり、GapFilling機能の肝心の「ギャップ埋め立て」が警戒中からライントレース中に戻る直前に行われる。この機能は、信号データを上書きする機能である。上書きする

⁴執筆時の値

対象となるのは警戒中に入ったところから、ライントレース中に戻ったところマイナス 26 ステップ。この範囲がギャップとされ、2 点が直線で結ばれる用にアルゴリズムが追跡マークを打ち、信号データを上書きする。この処理を行っている間はアルゴリズム状態は「埋め立て中」となり、追跡マークは青に変わる。埋め立て処理が終わった後、アルゴリズムはギャップ埋め立て処理直前の位置に戻り、状態は「ライントレース中」となる。

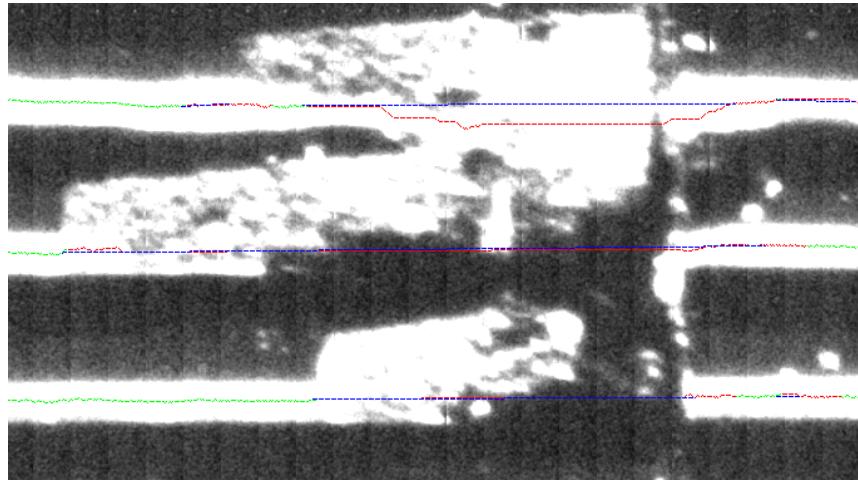


図 5.6: GapFilling 機能によるレコードに入っているひびで乱れたデータを修正

図 5.6 の画像データはレコードが完全に割れた画像データの場合の追跡マーク。青い点線が赤点線のデータを上書きする。これは GapFilling 機能が正常に動く例であり、アルゴリズムは乱れた音溝データの両端を直線で結び、データ修正が成功した。

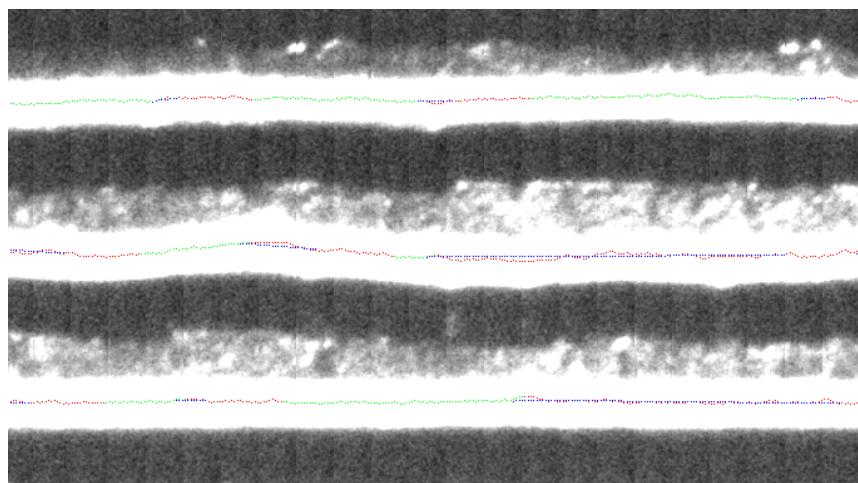


図 5.7: GapFilling 機能が正常な音声データを誤って修正した例

一方、図 5.7 は GapFilling 機能の判断が誤り、音声信号を誤って修正した例であ

る。これは音声データを悪化させたことであり、防ぎたいものの、ギャップ判定の判断基準を変えたりしない限り、GapFilling 機能の望ましからぬ副作用として目視せざるを得ない。

5.2.4 TryToRecover

図 5.8 の中心にある音溝データを見てみると赤い点線が、アルゴリズムの仕様の結果、黒領域に入ることが確認できる。いうまでもなく、一旦黒領域に入ると抜き出すことは困難であり、アルゴリズムは何万ステップも警戒中状態でとにかく真っすぐに進んでみる状態に陥る。こうなったときに、TryToRecover 機能が呼び出される。

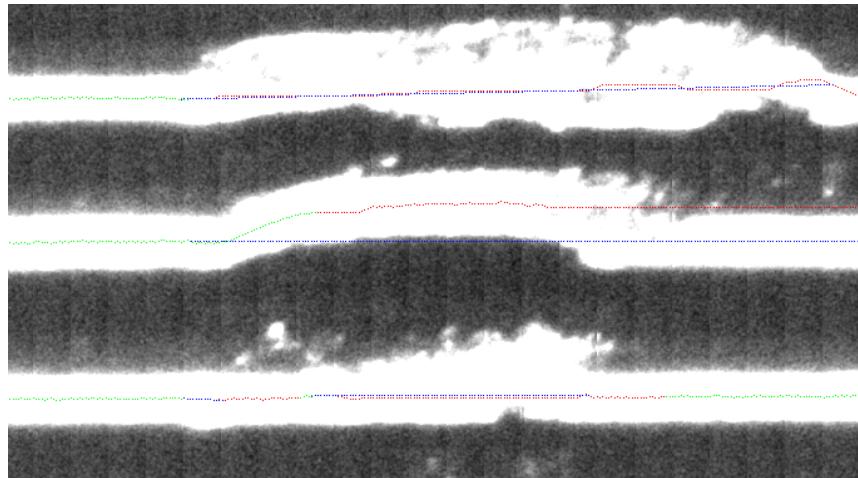


図 5.8: TryToRecover 機能によりアルゴリズムが無限警戒状態から回復

TryToRecover 機能の呼び出し条件は「黒領域（アルゴリズムの閾値以下のピクセル領域）を 600 ステップ以上連続的に進んだ」である。呼び出された後は現在縦座標値を 750 ステップ前（まだ黒領域ではなかった箇所）の値に設定し、その後 750 ステップ前の点と新しくできた座標点が直線で結ばれ、アルゴリズムが終点からライントレースを再開する。

5.3 Needle

5.4 fftf.m

Shmuel Ben-Ezra バンドができるように改造。

第6章 結論

付録A 物理的距離検出プログラム

A.1 PixelProcessing.m

```

1 % -----
2 % -- - Load Image Data - - -
3 % -----
4 tic
5 % load the original image
6 imgOriginal = imread('img/SmallP59000FL.bmp', 'bmp');
7 % load the original image
8 %imgOriginal = imread('img/P60400FL.bmp', 'bmp');
9 img = uint16(imgOriginal);
10 [imgHeight, imgWidth] = size(img(:,:,1));
11 %
12 % -- - GrayScale Bitmap - - -
13 %
14 grayImg = img(:,:,1) + img(:,:,2) + img(:,:,3);
15 grayImg = grayImg / 3;
16 imgNmbrOfPixels = length(grayImg(:));
17 %
18 % -- - - - - - - - - - - - - - -
19 % -- - - create TwoValImg - - -
20 %
21 threshold = 80;
22 TwoValImg = uint16(zeros(imgHeight, imgWidth));
23 % loop through all the pixels.
24 % The Pixels values go from 0 to 255*3
25 for pixel = 1:imgHeight*imgWidth
26     if grayImg(pixel) > threshold
27         TwoValImg(pixel) = 1;
28     else
29         TwoValImg(pixel) = 0;
30     end
31 end
32 %
33 %
34 %imageData = ImageDataClass(imgWidth, imgHeight);
35 %imageData.setup(grayImg, threshold);
36 %
37 % -- - - - - - - - - - - - - - -
38 % -- - - Show Results - - -
39 %
40 figure(1);

```

```

41 imagesc(imgOriginal);
42 axis image;
43
44 figure(2);
45 % display the image in matlab
46 imagesc(grayImg); colormap(gray);
47 axis image;
48
49 figure(3);
50 % display the image in matlab
51 imagesc(TwoValImg); colormap(gray);
52 axis image;
53
54 % - - - - - - - - - - - - - - - -
55 % - - - Pixel Processing - - - -
56 % - - - - - - - - - - - - - - - -
57 ObjectExtraction();

```

A.2 ObjectExtraction.m

```

1 X = TwoValImg;
2 [m,n] = size(X);
3
4 % - - - - - - - - - - - - - - - -
5 % - - - DELETE FRAME PIXELS - - -
6 % - - - - - - - - - - - - - - - -
7 X(1,:) = 0;
8 X(m,:) = 0;
9 X(:,1) = 0;
10 X(:,n) = 0;
11
12 % - - - - - - - - - - - - - - - -
13 % - - - - - - - LABELING - - - -
14 % - - - - - - - - - - - - - - - -
15 label = 0;
16 for kx = 2:m-1;
17     for ky = 2:n-1;
18         W = zeros(3);
19         W(1:3,1:3) = X(kx-1:kx+1, ky-1:ky+1);
20         maxW = max(W(:));
21         if X(kx,ky) == 1 && X(kx,ky-1) == 0 && maxW == 1;
22             label = label + 1;
23             X(kx,ky) = label;
24         end
25         if X(kx,ky) == 1 && maxW >= 1;
26             X(kx,ky) = maxW;
27         end
28     end
29 end
30 figure(4);
31 imagesc(X); axis image;
32 colormap(gray);
33

```

```

34 % -----
35 % - DELETE OVERLAPPING LABELS -
36 %
37 while 1
38     kk = 0;
39     for kx = 2:m-1;
40         for ky = 2:n-1;
41             if X(kx,ky) ~= 0 && X(kx+1,ky) ~= 0 && X(kx,ky) ~= X(kx+1,ky)
42                 old = X(kx,ky); new = X(kx+1,ky); kk = 1; break;
43             end
44         end
45     end
46     if kk == 0
47         break;
48     end
49     for kx = 2:m-1;
50         for ky = 2:n-1;
51             if X(kx,ky) == old;
52                 X(kx,ky) = new;
53             end
54         end
55     end
56 end
57 toc
58 figure(5);
59 imagesc(X); axis image
60 %
61 % -----
62 % - - - OBJECT EXTRACTION - - -
63 %
64
65 indexCounter = 0;
66 existingPixelObjects = [0];
67 PixelObjects = PixelObject_.empty(0, 0);
68 map = [];
69 for kx = 2:m-1;
70     for ky = 2:n-1;
71         tmp = X(kx, ky);
72         if tmp ~= 0;
73             ex = false;
74             for ent = existingPixelObjects;
75                 if ent == tmp;
76                     ex = true;
77                     break;
78                 end
79             end
80             if ex ~= 0;
81                 % The thing already exists!
82                 PixelObjects(map(tmp)).addPoint([kx, ky]);
83             else
84                 % The thing didn't exist!
85                 indexCounter = indexCounter + 1;
86                 existingPixelObjects(indexCounter) = tmp;
87                 % Here we need to add another entry to the map!

```

```
88         map(tmp) = indexCounter;
89         PixelObjects(indexCounter) = ...
90             PixelObject_(indexCounter);
91             PixelObjects(indexCounter).addPoint([kx, ky]);
92         end
93     end
94 end
95
96 % -----
97 % PRINT NMBR OF BIG PIXELOBJTS -
98 % -----
99
100 pixelThreshold = 1000;
101 nbrOfObjects = 0;
102
103 for ent = PixelObjects;
104     if ent.EntryIndex > pixelThreshold;
105         ent.EntryIndex
106         nbrOfObjects = nbrOfObjects + 1;
107     end
108 end
109 nbrOfObjects
```

付録B 画像処理プログラム

B.1 SpliceBot.m

```

1 % -----
2 % - - - Splice Bot Class - - -
3 % -----
4 classdef SpliceBot < handle
5     properties
6         CurrentRoundNmbr;
7         NmbrOfImages;
8         MaxFolderNmbr;
9         NmbrOfSplicedImages;
10    end
11    methods
12        %
13        % - - - Constructor - - -
14        %
15        function obj = SpliceBot()
16            obj.CurrentRoundNmbr = 1;
17            obj.GetNmbrOfFolders();
18        end
19        %
20        % - - - Start - - -
21        %
22        function Start(obj)
23            while obj.CurrentRoundNmbr <= obj.MaxFolderNmbr;
24                obj.GetNmbrOfFiles();
25                obj.AddImages();
26                obj.AddRemainingImages();
27                obj.GetNmbrOfSplicedImages();
28                obj.ConvertToGrayScale();
29                obj.ConvertToBWImages();
30                obj.Next();
31            end
32        end
33        %
34        % - - - Next - - -
35        %
36        function Next(obj)
37            obj.CurrentRoundNmbr = obj.CurrentRoundNmbr + 1;
38        end
39        %
40        % - - - Add Images - - -

```

```
41 % - - - - -
42 function AddImages(obj)
43     path = 'Round%d/rawdata/AIA%d.bmp'
44     path_ = 'Round%d/splicedImages/splicedImage%d.bmp'
45     splicedFullImages = floor(obj.NmbrOfImages/50);
46
47     for p = 0 : splicedFullImages-1;
48         tmp1 = imread(sprintf(path, ...
49                         obj.CurrentRoundNmbr,p*50 +1), 'bmp');
50
51         % crop the black pixels!
52         tmp1(end,:,:)= [];
53         tmp1(:,end,:)= [];
54         tmp1(1,:,:)= [];
55         tmp1(:,1,:)= [];
56
57         tmp1 = flipud(tmp1);
58         tmp1 = rot90(tmp1);
59         tmp1 = flipud(tmp1);
60
61         for i = p*50 +2 : (p+1)*50;
62             tmp = imread(sprintf(path,obj.CurrentRoundNmbr,i));
63             % crop the black pixels!
64             tmp(end,:,:)= [];
65             tmp(:,end,:)= [];
66             tmp(1,:,:)= [];
67             tmp(:,1,:)= [];
68
69             tmp = flipud(tmp);
70             tmp = rot90(tmp);
71             tmp = flipud(tmp);
72
73             tmp1 = cat(2,tmp1,tmp);
74         end
75         disp(p+1);
76         imwrite(tmp1,sprintf(path_,obj.CurrentRoundNmbr,p+1));
77     end
78 % - - - - -
79 % - - - AddRemainingImages - - -
80 % - - - - -
81 function AddRemainingImages(obj)
82     path = 'Round%d/rawdata/AIA%d.bmp'
83     path_ = 'Round%d/splicedImages/splicedImage%d.bmp'
84     splicedFullImages = floor(obj.NmbrOfImages/50);
85     rest = mod(obj.NmbrOfImages, 50);
86     if rest == 0
87         return;
88     end
89
90     tmp1 = ...
91         imread(sprintf(path,obj.CurrentRoundNmbr,splicedFullImages*50 ...
92                         +1), 'bmp');
```

```
92         % crop the black pixels!
93         tmp1(end,:,:)= [];
94         tmp1(:,end,:)= [];
95         tmp1(1,:,:)= [];
96         tmp1(:,1,:)= [];
97
98         tmp1= flipud(tmp1);
99         tmp1= rot90(tmp1);
100        tmp1= flipud(tmp1);
101
102        %for i = 10252 : 10285;
103        for i = splicedFullImages*50 +2 : obj.NmbrOfImages;
104            tmp = imread(sprintf(path,obj.CurrentRoundNmbr,i));
105            % crop the black pixels!
106            tmp(end,:,:)= [];
107            tmp(:,end,:)= [];
108            tmp(1,:,:)= [];
109            tmp(:,1,:)= [];
110
111            tmp = flipud(tmp);
112            tmp = rot90(tmp);
113            tmp = flipud(tmp);
114
115            tmp1 = cat(2,tmp1,tmp);
116
117        end
118        disp(splicedFullImages+1);
119        imwrite(tmp1,sprintf(path_,obj.CurrentRoundNmbr, ...
120                           splicedFullImages + 1));
121
122        % - - - - - - - - - - - - - - - - - -
123        % - - CONVERT TO GRayscale - - -
124        % - - - - - - - - - - - - - - - - - -
125        function ConvertToGrayScale(obj)
126            path = 'Round%d/rawdata/AIA%d.bmp'
127            for p = 1 : obj.NmbrOfSplicedImages;
128                img = imread(sprintf(path, obj.CurrentRoundNmbr, ...
129                               p), 'bmp');
130
131                grayImg = img(:,:,1) + img(:,:,2) + img(:,:,3);
132                grayImg = double(grayImg);
133                grayImg = grayImg / max(grayImg(:));
134                imgNmbrOfPixels = length(grayImg(:));
135                p
136                imwrite(grayImg,sprintf(path, obj.CurrentRoundNmbr, ...
137                                         p));
138
139        end
140
141        % - - - - - - - - - - - - - - - - - -
142        % - - - - - - - - - - - - - - - - - -
143        % - - - - - - - - - - - - - - - - - -
144        function ConvertToBWImages(obj)
145            path = 'Round%d/splicedImages/splicedImage%d.bmp'
146            path_ = 'Round%d/BWImages/BWImage%d.bmp'
```

```
143         for p = 1 : obj.NmbrOfSplicedImages;
144             img = imread(sprintf(path, obj.CurrentRoundNmbr, ...
145                         p), 'bmp');
146             bwImg = false(size(img));
147             bwImg(img > 250) = true;
148             p
149             imwrite(bwImg,sprintf(path_, obj.CurrentRoundNmbr, p));
150         end
151     % -----
152     % -- Get Number Of folders --
153     % -----
154     function GetNmbrOfFolders(obj)
155         D = dir(['.', '\Round*']);
156         obj.MaxFolderNmbr = length(D([D.isdir]));
157     end
158     % -----
159     % -- Get Number Of files --
160     % -----
161     function GetNmbrOfFiles (obj)
162         path = 'Round%d/rawdata'
163         D = dir([sprintf(path, obj.CurrentRoundNmbr), '\*.bmp']);
164         obj.NmbrOfImages = length(D(not([D.isdir])));
165     end
166     % -----
167     % -- Get Number Of spliced Img --
168     % -----
169     function GetNmbrOfSplicedImages (obj)
170         path = 'Round%d/splicedImages'
171         D = dir([sprintf(path, obj.CurrentRoundNmbr), '\*.bmp']);
172         obj.NmbrOfSplicedImages = length(D(not([D.isdir])));
173     end
174 end
175 end
```

B.2 SigExBot.m

```
1 % -----
2 % - - - SIG EX Bot CLASS - - -
3 %
4 classdef SigExBot < handle
5 properties
6     IsOnTrack;
7     FirstCycle;
8     TrackFollowing;
9     StepSize;
10    StartStepSize;
11    CurrentX;
12    CurrentY;
13    CurrentRoundNmbr;
14    ImgRGB = {};
15    Img = {};
16    ImgWidth;
17    ImgHeight;
18    PixelThreshold; % 0 ~ 255
19    CurrentImgNmbr;
20    MaxImgNmbr;
21    MaxImgNmbrFirstImgSet;
22    MaxRoundNmbr;
23    SignalIndex;
24    ProcessedSignal = [];
25    Signal = [];
26    DebugSignal1 = [];
27    SignalWidthArr = [];
28    MeanSignalWidth;
29    ChangeInSigWidth;
30    StrangeThingCounter;
31    Debug;
32    AlgoStopHeight;
33    CorVal;
34    CurrentCorVal;
35    GapFlag;
36    GapFillingFlag;
37    GapEndedCounter;
38    GapSlope;
39    StartOfGapX;
40    StartOfGapY;
41    StartOfGapIndex;
42    StartOfGapImgNmbr;
43    EndOfGapImgNmbr;
44    EndOfGapIndex;
45    EndOfGapY;
46    ReturnIndex;
47    ReturnX;
48    ReturnY;
49    TryToRecover;
50    TryToRecoverCounter;
51 end
```

```
52     methods
53         % - - - - - - - - - - - - - -
54         % - - - - - Constructor - - - -
55         % - - - - - - - - - - - - - -
56         function obj = SigExBot()
57             obj.TrackFollowing = false;
58             obj.FirstCycle = false;
59             % When the Bot is created, it isn't on track
60             obj.IsOnTrack = false;
61             obj.StartStepSize = 2.5;
62             obj.CurrentX = 1;
63             obj.CurrentY = 2400;
64             obj.ImgWidth = 0;
65             obj.ImgHeight = 0;
66             obj.PixelThreshold = 240;
67             obj.CurrentImgNmbr = 1;
68             obj.CurrentRoundNmbr = 1;
69             obj.StrangeThingCounter = 0;
70             obj.SignalIndex = 0;
71             obj.Signal(300000) = 0;
72             obj.ProcessedSignal(300000) = 0;
73             % Debug is On by default
74             obj.Debug = true;
75             % Algorithm stops when higher
76             % than 2000px at 1st image
77             obj.AlgoStopHeight = 2000;
78             % 1486px correction after one full Round
79             obj.CorVal = 1486;
80             obj.CurrentCorVal = 0;
81             obj.GapEndedCounter = 0;
82             obj.GapFlag = false;
83             obj.GapFillingFlag = false;
84             obj.TryToRecover = false;
85             obj.TryToRecoverCounter = 0;
86             obj.GetNmbrOfFolders();
87             obj.GetNmbrOfFiles();
88             obj.LoadImagesIntoRAM();
89             obj.ChangeCurrentImage(obj.CurrentImgNmbr);
90             obj.SetCurrentStepSize();
91         end
92         % - - - - - - - - - - - - - -
93         % - - Get Number Of folders - - -
94         % - - - - - - - - - - - - - -
95         function GetNmbrOfFolders(obj)
96             D = dir(['.', '\Round*']);
97             obj.MaxRoundNmbr = length(D([D.isdir]));
98         end
99         % - - - - - - - - - - - - - -
100        % - - Get Number Of files - - -
101        % - - - - - - - - - - - - - -
102        function GetNmbrOfFiles (obj)
103            D = dir([sprintf('Round%d/splicedImages', ...
104                  obj.CurrentRoundNmbr), '\*.bmp']);
105            obj.MaxImgNmbr = length(D(not([D.isdir])))
```

```

105         if ~obj.TrackFollowing;
106             obj.MaxImgNmbrFirstImgSet = obj.MaxImgNmbr;
107         end
108     end
109     % - - - - - - - - - - - - - - -
110     % -- Load Images into RAM -- -
111     % - - - - - - - - - - - - - -
112     function LoadImagesIntoRAM(obj)
113         for p = 1 : obj.MaxImgNmbr;
114             tmp = ...
115                 imread(sprintf('Round%d/splicedImages/splicedImage%d.bmp', ...
116                             obj.CurrentRoundNmbr, p), 'bmp');
117             obj.Img(p) = {tmp};
118             [m n r] = size(tmp);
119             tmpRGB = uint8(zeros(m,n,3));
120             tmpRGB(:,:,1) = uint8(tmp(:,:,1));
121             tmpRGB(:,:,2) = tmpRGB(:,:,1);
122             tmpRGB(:,:,3) = tmpRGB(:,:,1);
123             obj.ImgRGB(p) = {tmpRGB};
124         end
125     end
126     % - - - - - - - - - - - - - -
127     % -- Set Current StepSize -- -
128     % - - - - - - - - - - - - - -
129     function SetCurrentStepSize(obj)
130         obj.StepSize = obj.StartStepSize * obj.MaxImgNmbr / ...
131                         obj.MaxImgNmbrFirstImgSet;
132     end
133     % - - - - - - - - - - - - - -
134     % -- Go Back Images -- -
135     % - - - - - - - - - - - - - -
136     function GoBackImages(obj, nmbr)
137         tmp = obj.CurrentImgNmbr + nmbr;
138         if tmp < 1;
139             obj.CurrentImgNmbr = obj.MaxImgNmbr + tmp;
140         else
141             obj.CurrentImgNmbr = tmp;
142         end
143         if obj.Debug & obj.TrackFollowing;
144             disp(sprintf('Going Back one Image to: %d Image: ... ...
145                         %d', obj.CurrentRoundNmbr, obj.CurrentImgNmbr));
146         end
147         tmpSize = size(obj.Img{obj.CurrentImgNmbr});
148         obj.ImgHeight = tmpSize(1);
149         obj.ImgWidth = tmpSize(2);
150     end
151     % - - - - - - - - - - - - - -
152     % -- Change Current Image -- -
153     % - - - - - - - - - - - - - -
154     function ChangeCurrentImage(obj, nmbr)
155         if obj.Debug & obj.TrackFollowing;
156             disp(sprintf('processing Round: %d Image: %d', ...
157                         obj.CurrentRoundNmbr, obj.CurrentImgNmbr));
158         end

```

```
154     tmpSize = size(obj.Img{nmbr});
155     obj.ImgHeight = tmpSize(1);
156     obj.ImgWidth = tmpSize(2);
157     if (nmbr == 1 & (obj.ImgHeight - obj.AlgoStopHeight) > ...
158         obj.CurrentY & obj.TrackFollowing) | ...
159         (obj.CurrentRoundNmbr == obj.MaxRoundNmbr & ...
160          (obj.ImgHeight - obj.AlgoStopHeight) > ...
161          obj.CurrentY);
162     % even if there is a gap which
163     % the program tries to fill, stop it!
164     obj.GapFlag = false;
165     obj.SaveMarkedImages();
166     obj.CurrentRoundNmbr = obj.CurrentRoundNmbr + 1;
167     obj.CurrentCorVal = obj.CurrentCorVal - obj.CorVal;
168     obj.CurrentY = obj.CurrentY + obj.CorVal;
169     obj.GetNmbrofFiles();
170     obj.LoadImagesIntoRAM();
171     obj.SetCurrentStepSize();
172     if obj.CurrentRoundNmbr > obj.MaxRoundNmbr;
173         obj.TrackFollowing = false;
174     end
175     %
176     % - - - - - Next Step - - - - -
177     function NextStep(obj)
178         obj.CurrentX = obj.CurrentX + obj.StepSize;
179         obj.SignalIndex = obj.SignalIndex + 1;
180         obj.Signal(obj.SignalIndex) = obj.CurrentY + ...
181             obj.CurrentCorVal;
182         obj.DebugSignal1(obj.SignalIndex) = obj.ChangeInSigWidth;
183         if round(obj.CurrentX) > obj.ImgWidth;
184             obj.CurrentX = 1;
185             obj.CurrentImgNmbr = obj.CurrentImgNmbr + 1;
186             if obj.CurrentImgNmbr > obj.MaxImgNmbr;
187                 obj.CurrentImgNmbr = 1;
188             end
189             obj.ChangeCurrentImage(obj.CurrentImgNmbr)
190         end
191         %
192         % - - - - - Follow Track - - - - -
193         %
194         function FollowTrack(obj)
195             while obj.TrackFollowing;
196                 obj.CenterOnCurrentPos();
197                 obj.LeaveBlackMark();
198                 obj.NextStep();
199                 if obj.FirstCycle;
200                     obj.FirstCycle = false;
201                 end
202             end
203         end
```

```
204 % -----
205 % ----- Start -----
206 % -----
207 function Start(obj)
208     obj.TrackFollowing = true;
209     obj.FirstCycle = true;
210     if not(obj.IsOnTrack);
211         if not(obj.SearchClosestTrack());
212             'could not find closest Track!'
213             return
214         else
215             obj.FollowTrack();
216         end
217     else
218         obj.FollowTrack();
219     end
220 end
221 % -----
222 % ----- LeaveBlackMark -----
223 % -----
224 function LeaveBlackMark(obj)
225     if obj.Debug;
226         if obj.GapFillingFlag
227             obj.ImgRGB{obj.CurrentImgNmbr}(round(obj.CurrentY), ...
228                 round(obj.CurrentX), 1) = 0;
229             obj.ImgRGB{obj.CurrentImgNmbr}(round(obj.CurrentY), ...
230                 round(obj.CurrentX), 2) = 0;
231             obj.ImgRGB{obj.CurrentImgNmbr}(round(obj.CurrentY), ...
232                 round(obj.CurrentX), 3) = 255;
233         elseif obj.GapFlag
234             obj.ImgRGB{obj.CurrentImgNmbr}(round(obj.CurrentY), ...
235                 round(obj.CurrentX), 1) = 255;
236             obj.ImgRGB{obj.CurrentImgNmbr}(round(obj.CurrentY), ...
237                 round(obj.CurrentX), 2) = 0;
238             obj.ImgRGB{obj.CurrentImgNmbr}(round(obj.CurrentY), ...
239                 round(obj.CurrentX), 3) = 0;
240         else
241             obj.ImgRGB{obj.CurrentImgNmbr}(round(obj.CurrentY), ...
242                 round(obj.CurrentX), 1) = 0;
243             obj.ImgRGB{obj.CurrentImgNmbr}(round(obj.CurrentY), ...
244                 round(obj.CurrentX), 2) = 255;
245             obj.ImgRGB{obj.CurrentImgNmbr}(round(obj.CurrentY), ...
246                 round(obj.CurrentX), 3) = 0;
247         end
248     end
249 end
250 % -----
251 % ----- Save Marked Images -----
252 % -----
253 function SaveMarkedImages(obj)
254     obj
255     if obj.Debug;
256         disp('saving marked images');
257         for p = 1 : obj.MaxImgNmbr;
```

```
249         imwrite(obj.ImgRGB{p}, ...
250             sprintf('Round%d/markedImages/splicedImage%d.bmp', ...
251                 obj.CurrentRoundNmbr, p));
252     end
253 end
254 % - - - - - - - - - - - - - - -
255 % - - Get Centering Correction - -
256 % - - - - - - - - - - - - - - -
257 function val = GetCenterOfTrack(obj)
258     % How far up can we go?
259     % - - - - - - - - - - - - - -
260     incrementerUp = 0;
261     while(obj.Img{obj.CurrentImgNmbr}(round(obj.CurrentY) + ...
262         incrementerUp, round(obj.CurrentX)) >= ...
263             obj.PixelThreshold)
264         incrementerUp = incrementerUp - 1;
265     end
266     % How far down can we go?
267     % - - - - - - - - - - - - - -
268     incrementerDown = 0;
269     while(obj.Img{obj.CurrentImgNmbr}(round(obj.CurrentY) + ...
270         incrementerDown, round(obj.CurrentX)) >= ...
271             obj.PixelThreshold)
272         incrementerDown = incrementerDown + 1;
273     end
274
275     signalWidth = incrementerDown - incrementerUp;
276     obj.CalcMeanSignalWidth(signalWidth);
277
278     val = (incrementerUp + incrementerDown)/2;
279 end
280 % - - - - - - - - - - - - - -
281 % - - Center On Current Pos - - -
282 % - - - - - - - - - - - - - -
283 function CenterOnCurrentPos(obj)
284     if obj.GapFillingFlag == true;
285         obj.GapFilling();
286     else
287         if obj.Img{obj.CurrentImgNmbr}(round(obj.CurrentY), ...
288             round(obj.CurrentX)) >= obj.PixelThreshold;
289             obj.StrangeThingCounter = 0;
290             % How far up can we go?
291             % - - - - - - - - - - - - - -
292             incrementerUp = 0;
293             while(obj.Img{obj.CurrentImgNmbr}(round(obj.CurrentY) ...
294                 + incrementerUp, round(obj.CurrentX)) >= ...
295                     obj.PixelThreshold)
296                 incrementerUp = incrementerUp - 1;
297             end
298             % How far down can we go?
299             % - - - - - - - - - - - - - -
300             incrementerDown = 0;
301             while(obj.Img{obj.CurrentImgNmbr}(round(obj.CurrentY) ...
```

```

    + incrementeDown, round(obj.CurrentX)) >= ...
    obj.PixelThreshold)
    incrementeDown = incrementeDown + 1;
end

297     signalWidth = incrementeDown - incrementeUp;
298     obj.CalcMeanSignalWidth(signalWidth);

299     changeInY = (incrementeUp + incrementeDown)/2;

300     % make it so the max chaneInY is 1
301     if abs(changeInY) >= 1;
302         changeInY = changeInY/abs(changeInY);
303     end

304     obj.ResetTryToRecoverFlag();

305     % This is quite important:
306     % Here we check wether or not the signal
307     % width changed more than x (in this case 5)
308     % if this is the case, we declare
309     % the signal from here on as "gap".
310     % after getting back on track,
311     % the gap will be filled with a linear slope.
312     if obj.ChangeInSigWidth >= 5 & ~obj.TryToRecover;
313         if ~obj.GapFlag
314             % The code here runs only
315             % one cycle after gap was detected
316             obj.GapFlag = true;
317             obj.StartOfGapX = obj.CurrentX - 4 * ...
318                 obj.StepSize;
319             obj.StartOfGapIndex = obj.SignalIndex - 4;
320             if obj.StartOfGapX < 1;
321                 obj.StartOfGapX = obj.CurrentX;
322                 obj.StartOfGapIndex = obj.SignalIndex;
323             end
324             obj.StartOfGapY = ...
325                 obj.Signal(obj.StartOfGapIndex) - ...
326                     obj.CurrentCorVal;
327             obj.StartOfGapImgNmbr = obj.CurrentImgNmbr;
328         end
329         obj.GapEndedCounter = 0;
330         % as long as the change in signal width
331         % is bigger than x we go straight ahead.
332         changeInY = 0;
333     elseif obj.GapFlag;
334         % If we get in here, we are inside a gap,
335         % but the signal width has normalized to a ...
336             degree where
337             % Trackfollowing is possible again.
338             obj.GapEndedCounter = obj.GapEndedCounter + 1;
339             if obj.GapEndedCounter > 30;
340                 obj.GapFlag = false;
341                 obj.GapEndedCounter = 0;

```

```

342         obj.EndOfGapIndex = obj.SignalIndex - 26;
343         obj.ReturnIndex = obj.SignalIndex;
344         obj.ReturnX = obj.CurrentX;
345         obj.ReturnY = obj.CurrentY;
346         obj.EndOfGapY = ...
347             obj.Signal(obj.EndOfGapIndex) - ...
348                 obj.CurrentCorVal;
349             obj.CalculateSlope();
350             obj.GapFillingFlag = true;
351             obj.SignalIndex = obj.StartOfGapIndex;
352             obj.CurrentX = obj.StartOfGapX;
353             obj.CurrentY = obj.StartOfGapY;
354             obj.EndOfGapImgNmbr = obj.CurrentImgNmbr;
355             if ~(obj.StartOfGapImgNmbr == ...
356                 obj.CurrentImgNmbr);
357                 % Start and End on different Images
358                 obj.GoBackImages(obj.StartOfGapImgNmbr ...
359                                 - obj.CurrentImgNmbr);
360             end
361         end
362     end
363     obj.CurrentY = obj.CurrentY + changeInY;
364 else
365     % just go straight. Somethings strange happened!
366     % (We are in here because there is no pixel
367     % bright enough to do the centering on)
368     obj.StrangeThingCounter = ...
369         obj.StrangeThingCounter + 1;
370     if obj.StrangeThingCounter > 600;
371         % give we are already trying to recover and
372         % there is once again no bright pixelcluster
373         % to follow.
374         if obj.TryToRecover;
375             obj.SaveMarkedImages();
376             disp('Something strange happened!');
377             disp(obj.CurrentImgNmbr);
378             obj.TrackFollowing = false;
379         end
380         % after 600 steps of going blindly
381         % straight forward: Try to recover!
382         % We try to recover by setting the current Y
383         % value to the value 750 steps earlier
384         % (then everything should be still fine)
385         % Then we connect the X,Y point 750 steps ago
386         % with the current X and current (new) Y.
387         % We use a straight line for this.
388         obj.StrangeThingCounter = 0;
389         disp('Try to recover!');
390         obj.GapFlag = false;
391         obj.GapFillingFlag = true;
392         obj.TryToRecover = true;
393         obj.EndOfGapIndex = obj.SignalIndex;
394         obj.EndOfGapImgNmbr = obj.CurrentImgNmbr;
395         obj.ReturnIndex = obj.SignalIndex;

```

```
391         obj.ReturnX = obj.CurrentX;
392         obj.CurrentY = obj.SignalIndex(obj.SignalIndex - ...
393                                         750) - obj.CurrentCorVal;
394         obj.ReturnY = obj.CurrentY;
395         obj.EndOfGapY = obj.CurrentY;
396         obj.SignalIndex = obj.SignalIndex - 750;
397         obj.CurrentX = obj.CurrentX - 750 * ...
398                                         obj.StepSize;
399         obj.StartOfGapX = obj.CurrentX;
400         obj.StartOfGapY = obj.CurrentY;
401         obj.StartOfGapIndex = obj.SignalIndex;
402         obj.StartOfGapImgNmbr = obj.CurrentImgNmbr;
403         obj.CalculateSlope();
404         if obj.CurrentX < 1;
405             obj.GoBackImages(-1);
406             obj.StartOfGapImgNmbr = obj.CurrentImgNmbr;
407             obj.CurrentX = obj.ImgWidth + obj.CurrentX;
408         end
409     end
410 end
411 % - - - - -
412 % - - Reset Try To Recover Flag -
413 % - - - - -
414 function ResetTryToRecoverFlag(obj)
415     if obj.TryToRecover;
416         obj.TryToRecoverCounter = obj.TryToRecoverCounter + 1;
417         if obj.TryToRecoverCounter > 500;
418             obj.TryToRecoverCounter = 0;
419             obj.TryToRecover = false;
420         end
421     end
422 end
423 % - - - - -
424 % - - - GapFilling - - - -
425 % - - - - -
426 function GapFilling(obj)
427     if obj.SignalIndex < obj.EndOfGapIndex;
428         % do the gap filling!
429         obj.CurrentY = obj.StartOfGapY + (obj.SignalIndex - ...
430                                         obj.StartOfGapIndex) * obj.GapSlope;
431     else
432         % switch back to normal behavior
433         obj.GapFillingFlag = false;
434         obj.SignalIndex = obj.ReturnIndex;
435         obj.CurrentX = obj.ReturnX;
436         obj.CurrentY = obj.ReturnY;
437         obj.CurrentImgNmbr = obj.EndOfGapImgNmbr;
438         obj.ChangeCurrentImage(obj.CurrentImgNmbr);
439     end
440 end
441 % - - - - -
442 % - - - Calculate Slope - - -
```

```
442 % -----
443 function CalculateSlope(obj)
444     length = obj.EndOfGapIndex - obj.StartOfGapIndex;
445     height = obj.EndOfGapY - obj.StartOfGapY;
446     obj.GapSlope = height / length;
447 end
448 % -----
449 % -- Calc Mean Signal Width --
450 %
451 function CalcMeanSignalWidth(obj, sigWidth)
452     if obj.FirstCycle;
453         obj.MeanSignalWidth = sigWidth;
454         obj.ChangeInSigWidth = 0;
455     else
456         diff = sigWidth - obj.MeanSignalWidth;
457         if abs(diff) <= 18
458             obj.MeanSignalWidth = obj.MeanSignalWidth + ...
459                 diff/100;
460         end
461         obj.ChangeInSigWidth = abs(diff);
462     end
463 %
464 % -- Search closest Track --
465 %
466 function ret = SearchClosestTrack(obj)
467     for Ypixel = obj.ImgHeight : -1 : 1;
468         if obj.Img{obj.CurrentImgNmbr}(Ypixel, 1) >= ...
469             obj.PixelThreshold
470             if obj.PixelThreshold <= ...
471                 sum(sum(obj.Img{obj.CurrentImgNmbr}(Ypixel-15:Ypixel-5, ...
472                     1:100)) / 1000);
473                 obj.IsOnTrack = true;
474                 obj.CurrentY = Ypixel;
475                 ret = true; % Track found!
476                 return;
477             end
478         end
479     end
480     ret = false; % No track found!
481 end
482 %
483 % -- Process Signal --
484 %
485 function ProcessSignal(obj)
486     for entry = obj.Signal;
487         entry
488     end
489 end
490 end
```

B.3 Needle.m

```
1 % -----
2 % ----- Needle CLASS -----
3 %
4 classdef Needle < handle
5     properties
6         Signal;
7         TrendCorrectedSignal;
8         FFTOutput;
9         SmoothingOutput;
10        NeedleSimOut;
11        SignalLength;
12        Yn;
13        X;
14        SamplingRate;
15        Drag;
16        SignalLengthInSecs;
17        xTime;
18        %NeedleSimualtion properties
19        Speed;
20        Mass;
21        Acceleration;
22        NeedleForce;
23        SpeedResForce
24        SpringResForce
25    end
26    methods
27        %
28        % ----- Constructor -----
29        %
30        function obj = Needle(signal)
31            obj.Signal = signal;
32            obj.SignalLength = length(obj.Signal);
33            obj.Yn = [];
34            obj.Init();
35        end
36        %
37        % ----- Init -----
38        %
39        function Init(obj)
40            obj.Drag = 0.0001;
41            obj.X = linspace(1, obj.SignalLength, obj.SignalLength);
42            obj.SamplingRate = 180000;
43            obj.SignalLengthInSecs = obj.SignalLength / ...
44                obj.SamplingRate;
45            obj.xTime = linspace(0, obj.SignalLengthInSecs, ...
46                obj.SignalLength);
47        end
48        %
49        % ----- Start -----
50        %
51        function Start(obj)
```

```

50         obj.TrendCorrectSignal();
51         obj.FFT();
52         obj.Crop();
53         obj.AdjustAmplitude();
54
55         obj.NeedleSimulation();
56
57         obj.Plot();
58         obj.SaveWAV();
59     end
60
61 % - - - - - - - - - - - - - - -
62 % - - - NeedleSimulation - - - -
63 % - - - - - - - - - - - - - - -
64 function NeedleSimulation(obj)
65     obj.Acceleration = 0;
66     obj.Speed = 0;
67     obj.Mass = 2;
68     % Here we make another array
69     % the same size as the input signal
70     obj.NeedleSimOut(length(obj.FFTOutput)) = 0;
71     obj.NeedleSimOut(1) = obj.FFTOutput(1);
72     for i = 2:length(obj.FFTOutput);
73         % compute all the forces
74         % for the current step
75         obj.NeedleForce = obj.FFTOutput(i) - ...
76             obj.NeedleSimOut(i-1);
77         obj.SpeedResForce = -obj.Speed*1.0;
78         obj.SpringResForce = -obj.NeedleSimOut(i-1) * 0.1;
79
80         % a = (F + SpeedRes + SpringRes)/m
81         obj.Acceleration = (obj.NeedleForce + ...
82             obj.SpeedResForce + obj.SpringResForce)/obj.Mass;
83
84         % 0.01 because we don't want to add
85         % the full acceleration every time.
86         obj.Speed = obj.Speed + obj.Acceleration * 0.8;
87         obj.NeedleSimOut(i) = obj.NeedleSimOut(i-1) + ...
88             obj.Speed*0.09;
89
90         % First thing: The Force which tries
91         % to move the needle towards the Signal!
92         % -SignalForce
93
94         % Then: All the Forces which work
95         % against the first Force!
96         % -SpeedResistanceForce (the greater
97         % the moving speed the greater this Force)
98
99         % -SpringForce (tries to move
100            % needle back to center position)
101     end
102 end
103
104 % - - - - - - - - - - - - - - -
105 % - - - TrendCorrectSignal - - -

```

```

101      % -----
102      function TrendCorrectSignal(obj)
103          p = polyfit(obj.x, obj.Signal, 1);
104          obj.TrendCorrectedSignal = obj.Signal - polyval(p, obj.x);
105      end
106      % -----
107      % ----- SmoothingSignal -----
108      % -----
109      function SmoothingSignal(obj)
110          % Here we make another array the
111          % same size as the input signal
112          obj.Yn(length(obj.FFTOutput)) = 0;
113          obj.Yn(1) = obj.FFTOutput(1);
114          for i = 2:length(obj.FFTOutput);
115              diff = obj.FFTOutput(i) - obj.Yn(i-1);
116              obj.Yn(i) = obj.Yn(i-1) + obj.Drag * diff;
117          end
118          obj.SmoothingOutput = obj.FFTOutput - obj.Yn;
119      end
120      % -----
121      % ----- Plot -----
122      % -----
123      function Plot(obj)
124          figure('Name', 'FFTOutput + NeedleSimOut');
125          plot(obj.xTime, obj.FFTOutput);
126          hold on;
127          plot(obj.xTime, obj.NeedleSimOut);
128          %figure('Name', 'FFTOutput secs1');
129          %plot(obj.xTime, obj.FFTOutput);
130          %figure('Name', 'FFTOutput secs2');
131          %plot(obj.xTime, obj.FFTOutput);
132          %figure('Name', 'FFTOutput datapoints');
133          %plot(obj.FFTOutput);
134          %figure('Name', 'original Signal plot');
135          %plot(obj.Signal);
136      end
137      % -----
138      % ----- Adjust Amplitude -----
139      % -----
140      function AdjustAmplitude(obj)
141          obj.FFTOutput = obj.FFTOutput/ max(abs(obj.FFTOutput));
142      end
143      % -----
144      % ----- FFT -----
145      % -----
146      function FFT(obj)
147          %obj.TrendCorrectedSignal(1) = ...
148          %obj.TrendCorrectedSignal(length(obj.TrendCorrectedSignal));
149          [obj.FFTOutput, f, y, y2] = fftf(obj.xTime, ...
150                                         obj.TrendCorrectedSignal, 15000, 100); %15000,100
151      end
152      % -----
153      % ----- Crop -----
154      % -----

```

```
153         function Crop(obj)
154             % Crop the first and last
155             % entries of the FFTOutput
156             obj.FFTOutput(1:10000) = [];
157             obj.xTime(1:10000) = [];
158             len = length(obj.FFTOutput);
159             obj.FFTOutput(len-10000:len) = [];
160             obj.xTime(len-10000:len) = [];
161         end
162         % - - - - - - - - - - - - - - -
163         % - - - - - Save WAV - - - - -
164         % - - - - - - - - - - - - - - -
165         function SaveWAV(obj)
166             audiowrite('needleOutput.wav', obj.FFTOutput, ...
167                         obj.SamplingRate);
168             audiowrite('needleOutputSimuNeedle.wav', ...
169                         obj.NeedleSimOut, obj.SamplingRate);
170         end
170     end
```