イミュニティ試験における誤動作判断支援システムの開発

中村卓・木島一広・清水章良・河野裕

Development of Malfunction Judgement Support System in Immunity Testing

Takashi NAKAMURA, Kazuhiro KIJIMA, Akio SHIMIZU and Hiroshi KONO

要　約

電子機器の電磁両立性（EMC）に関する試験の一つに電磁波を印可して誤動作しないかを検査するイミュニティ試験があるが，供試体や試験条件によって誤動作の定義が異なる．そのため，試験者は試験の様子をカメラなどで見て誤動作を確認し続ける必要がある．本研究では，試験者の負担軽減と誤動作の見逃し防止を目的として，誤動作が疑わしい動作（非定常動作）を検出・通知するシステムの構築を行う．今年度は，非定常動作を検出するため，幾つかのケースにおける非定常動作検出アルゴリズムの作成を行った．作成したアルゴリズムを利用することで，カメラ映像から細かな非定常動作を検出することを確認した．

1.　緒　言

EMC に関する試験の一つに電磁ノイズを印可するイミュニティ試験がある．イミュニティ試験では，電磁ノイズを印可した際に電子機器が誤動作したか否かを試験者が判断する必要がある．誤動作の判断を行うために，試験者は試験中，常に供試体を観察し続ける，もしくは試験の様子を録画して試験後に確認するなどしなければならない．イミュニティ試験によっては一回の試験に長時間要し，それに伴い試験者も長時間観察し続ける必要があるため，試験者への負担も大きく，誤動作の見逃しも起こりうる．そのため，誤動作が疑わしい箇所を検出・提示するシステムの要望が，本センターでイミュニティ試験を行っている企業から多く寄せられている．

本研究では，イミュニティ試験補助のため，非定常動作を検出し，その結果を試験者に提示するためのシステムの構築を行う．非定常動作を検出することで，試験者の負担を軽減することができ，製品開発に関する作業効率の向上が期待される．また，小さな非定常動作も検出することが可能であるため，誤動作の見逃し防止や試験全体の精度向上に繋がる．今年度は，幾つかの非定常動作について，それらを検出するためのアルゴリズムの作成を行った．また，作成したアルゴリズムを用いて映像からの検出を試み，現在の課題などについても考察した．

2.　非定常動作

非定常動作を検出するにあたり，非定常動作を定義・分類する必要がある．また，分類した非定常動作毎に検出方法を変えていく必要があるため，検出方法についても検討を行った．

2－1　定常動作と非定常動作

非定常動作を分類していく上で，どのような動作を非定常動作とするかを定義する必要がある．そのために，正常時の動作や正常動作以外の動作，誤動作などを明確に分ける必要がある．それぞれの動作を以下のように定義し，それぞれを定常動作，非定常動作，誤動作と呼ぶこととした．また，各動作の関係については，図 1 のベン図のとおりである．

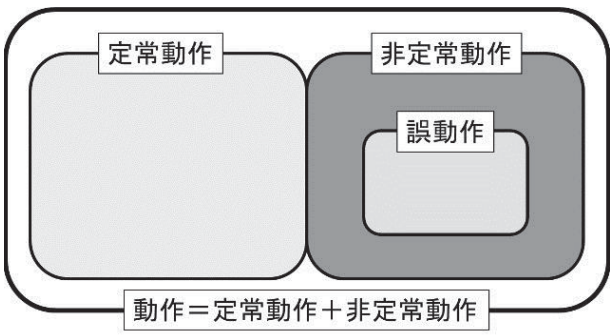

図 1　定常動作，非定常動作，誤動作の関係

表 1　非定常動作の種類と検出手法の例

非定常動作の種類	供試体での動作例	検出手法（例）	検出内容
表示の乱れ	・ プリンタ印字の乱れ ・ モニタ表示の乱れ	・フーリエ変換	・スペクトル
動作周期の乱れ	・ アンテナ回転周期の乱れ ・ LED 点滅間隔の乱れ	・フーリエ変換＋差分 ・オプティカルフロー	・時間の周期変化 ・移動ベクトル
数値の乱れ	・ 時計表示の乱れ ・ 計測器の値が大きく動く	・文字認識	・テキスト

・定常動作　：ノイズ未印可時の動作
・非定常動作：定常動作以外の動作
・誤　動　作：試験者が非定常動作を観察して
　　主観的にエラーと判断した動作

誤動作は，同じ供試体でも想定する動作環境などで誤動作の判断内容が異なるため，誤動作を機械的に検出する場合，試験機器の動作のみでなく環境なども考慮したうえで検出方法を作成しなければならない．また，誤動作か否かの最終的な判断は試験者の主観が必要となるため，誤動作のみを機械的に検出することは難しい．

非定常動作は，定義より図 1 のように誤動作も内包しているため，誤動作よりも幅広く捉えることができる．また，定常動作以外の動作を捉えればよく，非定常動作も試験条件などを考慮する必要がないため，誤動作よりも機械的に捉えやすい．そのため，本研究では，非定常動作の検出・提示までを行い，誤動作か否かの最終的な判断は試験者の主観に委ねることにした．非定常動作を検出するために，非定常動作の種類などの調査を行い，検出アルゴリズムの検討を行った．

2－2　非定常動作の分類

2－1でも述べた通り，誤動作は主観的な判断が必要となるため，誤動作を分類して，その種類ごとに検出手法を作成することは非常に難しい．非定常動作は誤動作と比べて機械的な検出は容易だが，より効率よく検出を行うためには，非定常動作の種類を分類して，それぞれに合った検出手法を実装する必要がある．本年度は，非定常動作の中でも本センターでの試験時に多く見受けられる“表示の乱れ”・“動作周期の乱れ”・“数値の乱れ”の3種類について検出を試みた．それぞれの非定常動作の特徴や検出方法の例について表1に示す．

3.　非定常動作の検出アルゴリズム

前述の 2－2 で検出対象とした 3 つの非定常動作について，検出アルゴリズムの作成を試みた．

3－1　検出アルゴリズムの流れ

それぞれの非定常動作の検出方法について説明する前に，全体の検出方法の流れについて説明する．いずれの検出アルゴリズムも定常動作時の動作と試験中の動作との比較を行い，その結果が定常動作時と異なっていると判断した場合，非定常動作と判断する．大まかな検出の流れについては以下のとおりである．

1. 定常動作時の映像から動作の特徴を抽出して，テンプレートを作成（特徴抽出は，検出対象の動作に応じて画像処理や文字認識などを利用）
2. 試験映像から 1. と同じ手法で特徴を抽出
3. 上記の 1. と 2. を比較して異なる場合は非定常動作と判断してその結果を提示

最終的に誤動作か否かの判断は，試験者が 3. で提示された結果を見て判断を行う．

3－2　表示の乱れの検出アルゴリズム

表示の乱れの検出方法は，画素ごとに時系列でフーリエ変換を行い，その結果を比較することで非定常動作の検出を行う．アルゴリズムの流れは以下のとおりである（図 2 参照）．

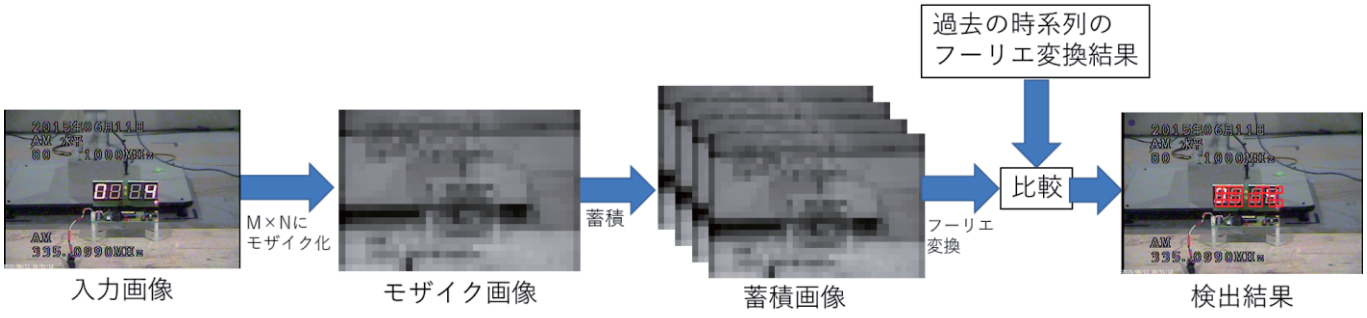

図 2　表示の乱れの検出の流れ

1. 入力映像から 1 フレームを取得し，グレースケール化した後に，M×N にモザイク化
2. モザイク化した画像を k フレーム蓄積
3. 蓄積した画像を画素毎に時系列でフーリエ変換
4. 1 フレーム前までの画像でフーリエ変換した結果を画素毎に比較

アルゴリズム中の M・N・k については，供試体に合わせて変更する．4．の比較方法については，フーリエ変換結果の 2 番目に大きい要素と最大値の要素を除く要素との平均値で行っている．最大値は画像の変化による差がほとんど見受けられない一方で，2 番目に大きい要素は画像の変化の影響を受けやすいため，最大値の要素は除外している．図 3 は検出アルゴリズムを適用した結果である（図中の点線で囲まれた箇所を非定常動作と認識）．

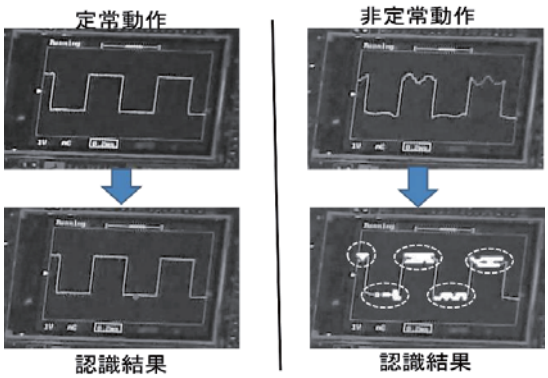

図 3　表示の乱れの検出結果

3−3　動作周期の乱れの検出アルゴリズム

“動作周期の乱れ”については，モータ（扇風機）の回転速度の変化の検出を試みた．基本的なアルゴリズムは“表示の乱れ”の場合と同じである．しかし，動作周期の場合には，1．のモザイク化の後に直前のフレームをモザイク化した画像との絶対値の差分を取り，0 から 1 にノーマライズをした画像を k フレーム蓄積して，フーリエ変換を行っている．また，回転箇所の差分が大きくなるように図 4 のようなマーカを供試体の回転部分に取り付け，回転箇所の時間変化を捉えやすくした．

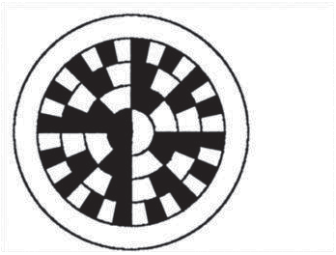

図 4　取り付けたマーカ

回転速度が一定の場合，ある箇所の時間変化は周期的になるため，フーリエ変換結果で前後を比較しても差が発生しない．回転速度が変化している場合，時間変化が非周期的になるため，前後の結果を比較した場合，差がみられる箇所が増加する傾向にある．停止いている場合，画像のフレーム間差分をとった場合，照明などによる輝度変化しか捉えないため，差分結果をノーマライズした場合，画像の中で変化が大きい（＝値が 1 に近くなる）箇所が全体に発生する．その結果，時間変化が非周期的な画素が非常に多くなる．そのため，図 5 のように時間変化が非周期的な画素数から定常動作（速度が一定）か非定常動作（速度が変化，動作停止）かを判断することができる．

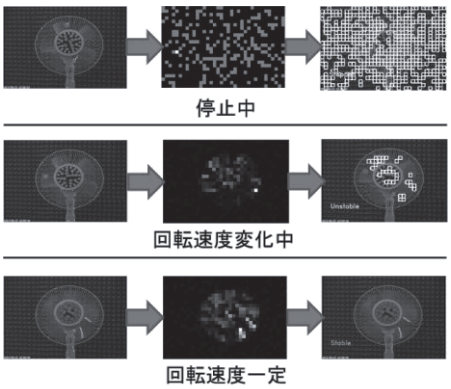

図 5　動作周期の乱れの検出結果（左：入力画像，中：フレーム間の差分（ノーマライズ後），右：検出結果（白い箇所が時間変化が非周期的な画素））

3−4　数値の乱れの検出アルゴリズム

“数値の乱れ”を検出するために，まず文字認識により数値を読み取る必要がある．そのため，図 6 のように試験映像から必要な領域を取り出し，下処理（二値化，アフィン変換，膨張・収縮処理など）を行い，数値の部分のみを切り出す．その後，光学文字認識（OCR）を用いて文字認識を行い，数値を読み取る．読み取った数値と予め定めたルール（例：カウントアップ（ダウン），数値が定めた範囲内にある，など）と照合し，ルールから外れた場合は非定常動作と判断する．

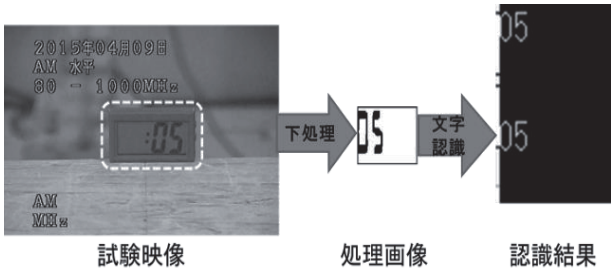

図 6　数値の乱れの検出の流れ

3－5　実装環境

実装は，一台のデスクトップ PC（Windows10，CPU: Core-i7-6700, RAM: 32 GB, GPU: NVIDIA Quadro M2000）を利用し，VC++（Visual Studio 2015）上で OpenCV3.2[1] および，CUDA7.0[2]を用いて実装を行った．文字認識にはオープンソース OCR の Tesseract[3] を利用した．試験映像は，720×480 ピクセル，30 fps で予め撮影したものを動画ファイルとして読み込んだ．アルゴリズムは，CPU と GPU の 2 種類で作成を試みた．

4. 考　察

第 3 章で作成した各アルゴリズムについて，実際の試験映像を用いて非定常動作を正しく検出できるか確認を行い，現状の課題などを検証した．また，CPU と GPU による処理速度の違いなどについても考察した．

4－1　検出アルゴリズムの精度

作成した各検出アルゴリズムについて，試験中の動画ファイルを用いて検出精度や現在の課題などについて考察した．“表示の乱れ”と“動作周期の乱れ”については，非定常動作を概ね捉えることができ，“動作周期の乱れ”は当初想定していなかった停止状態を検出することが可能であった．“表示の乱れ”については，モザイクサイズの大きさや蓄積数によって映像中の乱れが正しく検出されないケースがあったため，供試体に応じてこれらのパラメータを調整する必要がある．

“数値の乱れ”については，他の 2 つのアルゴリズムと異なり，検出精度がやや悪く，正しく数値を読み取れないことが多かった．原因としては，下処理を試験映像に応じて細かく調整する必要があることや，下処理の結果により OCR による文字認識結果が異なることがあったためと考えられる．解決策として，数値など検出する文字が限定されている場合には，OCR による文字認識ではなく，文字などを予め画像テンプレートとして用意してテンプレートマッチングを行い，その結果から文字認識を行うなどの方法が考えられる．“数値の乱れ”については，今後もアルゴリズムの改良が必要である．

4－2　CPU と GPU による処理の違い

CPU と GPU を用いて各アルゴリズムの実装を行い，“表示の乱れ”の検出アルゴリズムの処理速度（fps：1 秒当たりの処理フレーム数）の計測を行った．その結果を表 2 に示す．パラメータは，表 2 のようにモザイクサイズの大小（大：20×20 ピクセル，小：5×5 ピクセル）と蓄積数の多少（多：32 フレーム，少：16 フレーム）を変化させ，全 4 パターンで計測を行った．なお，使用した映像は 3 分 2 秒（6063 フレーム，30 fps）で映像の全領域（720×480 ピクセル）を走査対象とした．

表 2　パラメータによる処理速度の違い

モザイクサイズ	蓄積数	処理速度 (CPU, fps)	処理速度 (GPU, fps)
20×20（大）	16（少）	231.7	128.3
20×20（大）	32（多）	193.7	115.6
5×5（小）	16（少）	55.12	57.64
5×5（小）	32（多）	27.4	37.9

その結果，モザイクサイズが小さく蓄積数が多い場合には，GPU を用いて処理するほうが速い．一方，モザイクのサイズが大きく蓄積数が少ない場合には，メモリ転送などの問題から CPU の方が速いが，CPU による処理は検出条件が厳しくなるとリアルタイム処理（＝処理速度が 30 fps 以上）が難しくなる恐れがある．

パラメータによる処理速度の違いから，検出条件や走査領域などの設定によってどちらを利用するのか適切であるかの検討を行う必要がある．また，試験中に非定常動作の検出を行うことを想定した場合には，リアルタイム性を考慮する必要があるため，GPU による処理の方が適切であると思われる．

5. 結　言

イミュニティ試験における誤動作判断を支援するために，非定常動作を検知・通知するシステムの開発を行っている．本年度は，いくつかのケースについて非定常動作検出アルゴリズムを作成し，現在の課題などについて考察を行った．

次年度は，検出アルゴリズムの精度向上や検出可能な非定常動作の種類の増加に取り組んでいく予定である．また，アルゴリズムの改良以外に，試験者に非定常動作を通知するシステムを構築し，構築したシステムを本センターでイミュニティ試験を行う企業などに利用してもらえるようにする予定である．

参考文献

1）　OpenCV <http://opencv.org/>（2016-10-13 参照）

2）　CUDA <https://developer.nvidia.com/cuda-toolkit>（2017-1-15 参照）

3）　Tesseract <https://github.com/tesseract-ocr>（2016-10-20 参照）