Lecture d'affichages d'instruments de mesure par logiciel neuronal et gestion des résultats

Ing. B. MAYNÉ
Ir F. GUEUNING
ECAM Bruxelles

Ce travail consiste en l'élaboration d'un système d'acquisition par caméra TCP/IP d'afficheurs d'instruments de mesure (afficheur type : multimètre de laboratoire), grâce à un programme en C# utilisant le principe du système de neurones artificiels pour la reconnaissance. Une gestion des résultats est faite par un traitement de données afin de vérifier la reconnaissance ainsi que récupérer les dix valeurs les plus stables de l'acquisition pour toutes les caméras.

Mots-clefs : Système d'acquisition, Afficheurs d'instruments de mesure, Caméra TCP/IP, C#, Reconnaissance de caractères, Système de neurones artificiels

This paper consists in the elaboration of an acquisition system displaying measuring instrument (display : lab multimeter) by TCP/IP camera using a C# program and the artificial neuron network for the character recognition. A data processing manages the results in order to check the recognition as well as to collect the ten most steady data of the acquisition for all the cameras.

Keywords : Acquisition system, Displaying measuring instrument, TCP/IP camera, C#, Character recognition, Artificial neural network

1. Introduction

Ce projet a été réalisé chez LBT Testing & Calibration (à Louvain-La-Neuve), une entreprise spécialisée dans l'étalonnage de sondes en température et humidité, dans le but d'automatiser la prise de mesure.

Les appareils envoyés par le client afin d'en réaliser l'étalonnage possèdent différentes interfaces ou ports de communication, ce qui rend difficile une acquisition similaire pour tous. Parfois, certains appareils ne possèdent même qu'un simple afficheur.

Le but de ce travail est de réduire au minimum l'intervention directe d'un technicien de laboratoire durant le processus d'étalonnage (celui-ci pouvant durer jusque dix heures). Grâce à l'automatisation, le technicien peut ainsi s'impliquer dans d'autres tâches tout en gardant un oeil sur le déroulement du processus. De même, ce système permet de diminuer le risque d'erreur humaine.

Afin de remédier à cela, l'entreprise a souhaité créer un programme de lecture d'afficheurs d'instruments de mesure qui rendrait automatique et systématique la lecture de tous les appareils clients. Pour cela, les principes de reconnaissance de caractères et du système de neurones artificiels ont été utilisés.

2. Présentation du projet

Ce projet consiste en l'élaboration d'un système de reconnaissance de caractères sur des images provenant de caméras, prises avec un intervalle de trente à soixante secondes. Le système par caméra est le plus universel, car comparable aux yeux du technicien de laboratoire. L'interface retenue pour ces caméras est le protocole TCP/IP. Ce type de communication est actuellement le plus moderne et le plus utilisé. De plus, il permet de travailler à distance de l'entreprise, grâce à une simple connexion internet. Un logiciel écrit en C# s'occupe de l'acquisition et de la communication avec les caméras, ainsi que de l'interface avec l'utilisateur.

Les afficheurs que le programme doit pouvoir reconnaitre sont, par exemple,

des multimètres de laboratoire, des afficheurs de sonde, etc. Ils sont de couleurs différentes, de tailles différentes, certains possèdent plusieurs lignes de caractères avec, parfois, des graphiques. Les afficheurs peuvent être de type 7-segments ou de type pixel. Seules les valeurs numériques sont reconnues, et représentent la mesure prise par l'afficheur.

La reconnaissance de caractères (OCR) se fera grâce au système de neurones artificiels (système neuronal), issu de l'intelligence artificielle. Ce principe est de plus en plus utilisé dans les systèmes complexes modernes de recherche de données, de reconnaissance vocale ou de reconnaissance de formes et de caractères.

Pour finir, un système de vérification des données et de la reconnaissance est mis en place, afin de récupérer les dix valeurs les plus stables des quatre caméras nécessaires pour l'étalonnage.

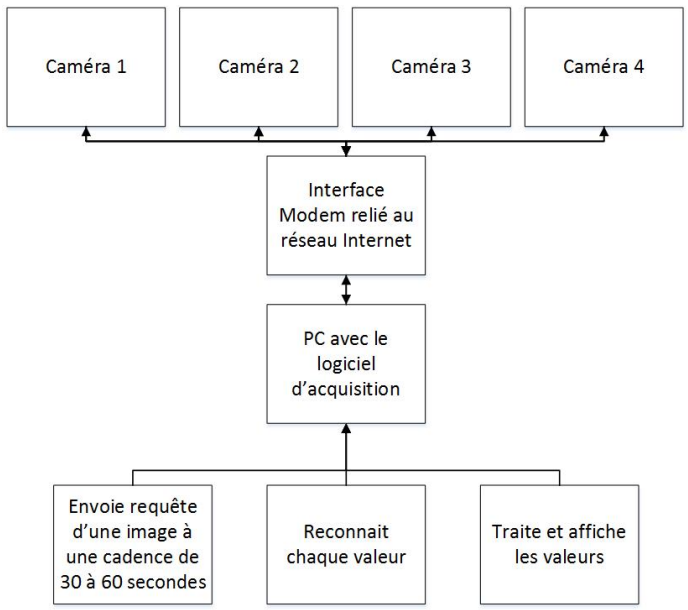

Figure 1: Principe Général

Dans cet article seules les parties techniques sont présentées. La figure 1 illustre le principe général alors que la figure 2 expose le cahier des charges tel qu'il a été décidé.

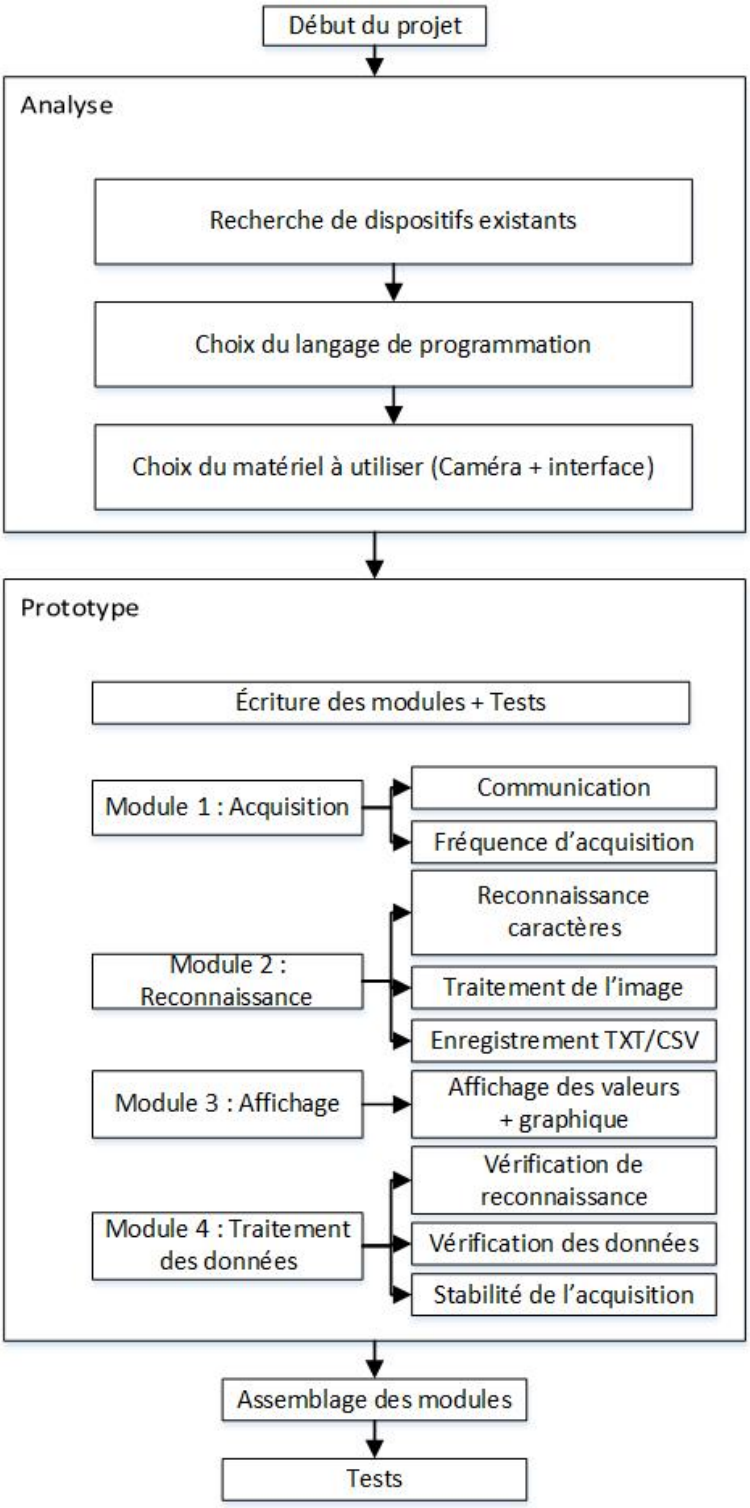

Figure 2: Cahier des charges

3. Préétude

3.1. Acquisition

Au vu de la difficulté d'intégrer un logiciel existant, l'écriture en C# d'un nouveau logiciel pour l'acquisition a été choisie. Pour communiquer avec la caméra TCP/IP, il suffit d'envoyer à la caméra, via son adresse IP, une requête HTTP spécifique au modèle de la caméra. Dans ce cas-ci, la caméra choisie est le modèle Zavio F1105 [7]. Celui-ci est simple, fonctionnel et bon marché. Il possède une résolution HD de 1280x720 et supporte le Wifi ainsi que le format MJPEG, qui est un format entre le Jpeg (format image) et le MPEG (codec vidéo). La caméra filme en continu, en images prédécoupées. En envoyant la requête, elle renvoie, sous forme d'une matrice de bits, l'image actuelle. Cette image est ensuite affichée et enregistrée sur le disque comme sauvegarde.

Le choix du langage de programmation C# a été favorisé grâce à l'environnement exclusif de l'entreprise qui est sous Windows. Une grande communauté existe également sur le net, en plus de posséder une large bibliothèque gérant les entrées/sorties, le temps réel, les requêtes par paquet, etc. Il s'agit d'un langage objet de haut niveau présentant beaucoup de similarité avec le Java.

La figure 3 illustre un exemple de code servant à l'acquisition.

3.2. Reconnaissance par le système neuronal

Pour la reconnaissance, à la vue des contraintes liées aux nombreux types d'afficheurs existants, le système de neurones artificiels a été choisi. Ce système provient de l'intelligence artificielle, est très complexe mais très efficace. D'autres systèmes ont été testés sans donner de résultats suffisants, comme Tesseract [5] de Google.

La principale qualité de ce système est sa capacité à reconnaitre des caractères et a en déduire la solution la plus probable, après ce qu'on appelle un entrainement.

Le principe du système de neurones artificiels est exposé dans la figure 4. Le système possède trois couches : une couche d'entrée, une couche intermédiaire et une couche de sortie. La couche d'entrée est celle où se retrouvera

```
1  //Déclaration d'un buffer pour la réception de l'image
2  byte[] buffer = new byte[100000000];
3  //Déclaration et initialisation des variables
4  int read, total = 0;
5  //Url de la caméra avec son adresse IP et son port
6  string sourceURL = "http://192.168.0.83:80/cgi-bin/view/image";
7  //Création de la requête HTTP
8  HttpWebRequest req = (HttpWebRequest)WebRequest.Create(sourceURL);
9  //Envoi du login et mot de passe à la caméra
10 req.Credentials = new NetworkCredential("utilisateur", "mdp");
11 //Reçoit une réponse
12 WebResponse resp = req.GetResponse();
13 //Reçoit une réponse sous forme de flux à lire
14 Stream = resp.GetResponseStream();
15 //Lecture des données
16 while ((read = stream.Read(buffer, total, 1000)) != 0)
17   {
18     total += read;
19   }
20 //Acquisition de l'image d'après les données reçues de la caméra
21 Bitmap bmp = (Bitmap)Bitmap.FromStream(new MemoryStream(buffer, 0, total));
22 //Test afin de mettre l'image dans la PictureBox en proportion
23 if (bmp.Width > pictureBox.Width)
24   pictureBox.SizeMode = PictureBoxSizeMode.StretchImage;
25 else
26   pictureBox.SizeMode = PictureBoxSizeMode.Normal;
27 //Mise de l'image dans la PictureBox
28 pictureBox.Image = bmp;
29 //Sauvegarde de l'image dans le dossier avec comme nom la date et l'heure à
       la seconde près
30 bmp.Save(DateTime.Now.ToString("yyMMdd hhmmss") + ".jpg",
       System.Drawing.Imaging.ImageFormat.Jpeg);
```

Figure 3: Exemple de code pour l'acquisition vidéo d'une caméra IP

l'image du caractère à reconnaitre. La couche intermédiaire contiendra la bibliothèque, composée d'images représentant les chiffres de 0 à 9 et prises par la caméra pour servir de référence lors de la reconnaissance de caractères. Enfin, la couche de sortie contiendra l'image que le système aura reconnue. Les trois couches et leurs éléments sont interconnectés pour permettre le bon déroulement du processus.

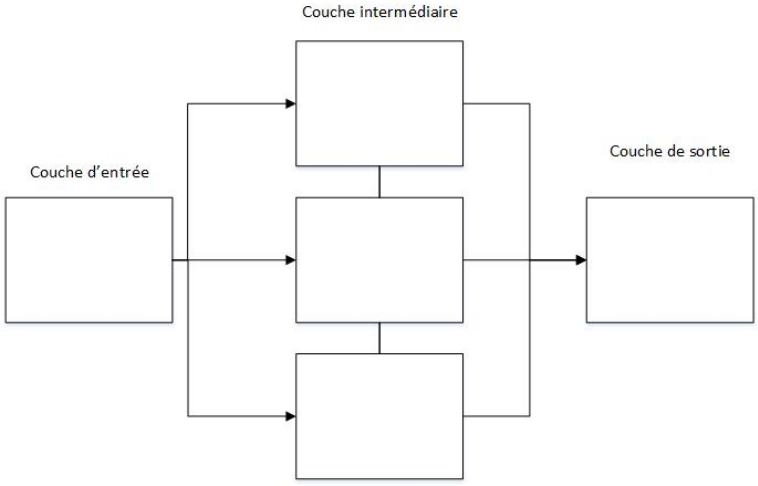

Figure 4: Principe du système neuronal

Jusqu'ici, le système présenté est un système par comparaison. La spécificité des neurones est que chaque connexion possède un poids, compris entre 0 et 1. Ces poids représentent la probabilité que l'image reliée avec cette connexion soit la bonne. Par exemple, si les poids sont correctement ajustés, le poids de la connexion entre une image représentant un 0 et l'image représentant un 0 de la bibliothèque serait de 1, alors que les connexions vers les autres chiffres seraient proches de 0. Cela représente le fait que le chiffre 0 est la solution la plus probable lorsque l'on tente de reconnaitre le chiffre 0 en entrée. La figure 5 montre le code pour le formatage des données des couches.

La documentation utilisée pour l'élaboration de ce projet a été trouvée sur le site [3] (le contenu est open-source et libre d'utilisation). Les articles consultés étaient accompagnés d'un code source "exemple" d'environ dix mille lignes écrit en C#. Chaque article présentait une explication d'une partie du système ainsi que de commentaires du code source. Voici les quatre articles étudiés : *« Image Recognition with Neural Networks »* [4] de Murat

Firat, *« Unicode Optical Character Recognition »* [1] de Daniel Admassu, *« Creating Optical Character Recognition (OCR) applications using Neural Networks »* [2] dAlex Cherkasov et enfin *« Hopfield model of neural network for pattern recognition »* [6] de Bashir Magomedov. Le programme final, utilisant le système neuronal, provient en partie d'un mélange et d'une adaptation de ces codes sources donnés. Ils ont servi de base à l'utilisation de ce système. Pour des explications plus poussées sur celui-ci, le lecteur pourra se référer aux liens fournis ci-dessus.

```
31 //Structure de la couche de pré entrée
32 struct PreInput
33 {
34   public double Value;
35   public double[] Weights;
36 };
37 //Données à sérialiser
38 [Serializable]
39 //Structure de la couche dentrée
40 struct Input
41 {
42   public double InputSum;
43   public double Output;
44   public double Error;
45   public double[] Weights;
46 };
47 //Données à sérialiser
48 [Serializable]
49 //Structure de la couche intermédiaire
50 struct Hidden
51 {
52   public double InputSum;
53   public double Output;
54   public double Error;
55   public double[] Weights;
56 };
57 //Données à sérialiser
58 [Serializable]
59 //Structure pour la couche de sortie
60 struct Output<T> where T : IComparable<T>
61 {
62   public double InputSum;
63   public double output;
64   public double Error;
65   public double Target;
66   public T Value;
67 };
```

Figure 5: Code du stockage des données pour les couches

Pour utiliser ce système, il est d'abord nécessaire de créer une bibliothèque d'images classées en dix groupes, correspondant aux chiffres de 0 à 9.

Ensuite, chaque image subit un traitement de l'image particulier (figure 6). L'image en couleur, sous forme d'une matrice de pixels, est convertie en noir et blanc. Cela permet de rendre le système insensible à la couleur qui pourrait différer d'une image à l'autre à cause de la luminosité. Pour ce faire, une moyenne de la couleur des pixels est réalisée. Ensuite, chaque valeur de pixel au-dessus de la moyenne est ramenée à du noir, en dessous à du blanc, sous forme de 1 pour le noir et de 0 pour le blanc. Cela permet de se retrouver avec une matrice de 0 et 1, facilement utilisable par le système.

```
68 //Fonction de conversion dune image RGB en matrice de bit
69 public static double[] Matrice(Bitmap BMP, int NombreLigneMatrice, int
       NombreColonneMatrice)
70 {
71   //Déclaration de variables doubles simples et tableaux
72   double ProportionHauteur = ((Double)NombreLigneMatrice / BMP.Height) ;
73   double ProportionLargeur = ((Double)NombreColonneMatrice / BMP.Width) ;
74   double[] Resultat = new double[NombreColonneMatrice * NombreLigneMatrice];
75   //Balaye chaque ligne de limage
76   for (int a = 0 ; a < NombreLigneMatrice ; a++)
77     {
78       //Balaye chaque colonne de limage
79       for (int b = 0 ; b < NombreColonneMatrice ; b++)
80       {
81         //Met chaque couleur dans la matrice de pixel
82         Color Couleur = BMP.GetPixel((int)(b / ProportionLargeur), (int)(a /
               ProportionHauteur)) ;
83         //Sort les résultats de la conversion
84         Resultat[a * NombreColonneMatrice + b] = 1 - (Couleur.R * .3 +
               Couleur.G * .59 + Couleur.B * .11) / 255 ;
85       }
86     }
87   //Retourne le résultat de la fonction
88   return Resultat ;
89 }
```

Figure 6: Code de la conversion en N/B

Disposant maintenant d'une bibliothèque, un entrainement du système va être effectué. Pour cela, on utilisera les fonctions "ForwardPropagate" (figure 7) et "BackPropagate" (figure 8) propre à l'intelligence artificielle. Elles représentent la propagation des données vers l'avant et l'arrière. Pour la reconnaissance, seule la fonction "ForwardPropagate" est utilisée, on part de l'entrée pour aller vers la sortie. Pour l'entrainement, comme expliqué ci-dessous, les deux fonctions sont utilisées, car la reconnaissance est exécutée, mais l'erreur de sortie est utilisée pour ajuster les poids à l'entrée.

Afin d'effectuer l'entrainement (figure 9), les poids des liaisons sont d'abord

```
90  //Fonction de propagation des données pour l'entrainement
91  public void ForwardPropagate(double[] pattern, T output)
92  {
93    //Déclaration d'un double
94    double total = 0.0 ;
95    //Déclaration d'un entier
96    int i, j ;
97    //Application des entrées au réseau actuel
98      for (i = 0 ; i < PreInputNum ; i++)
99      {
100       //On reprend la valeur du nombre déléments pour le set d'entrainement
101       PreInputLayer[i].Value = pattern[i] ;
102     }
103   //On calcule les entrées, les sorties et les erreurs
104   for (i = 0 ; i < OutputNum ; i++)
105   {
106     //On initialise la valeur de la variable
107     total = 0.0 ;
108     //Pour chaque élément de 0 au nombre dentrées
109     for (j = 0 ; j < PreInputNum ; j++)
110     {
111       //On incrémente le total en multipliant le poids et la valeur de la
              couche d'entrée
112       total += PreInputLayer[j].Value * PreInputLayer[j].Weights[i] ;
113     }
114     //On place les variables en sortie
115     OutputLayer[i].InputSum = total ;
116     OutputLayer[i].output = F(total) ;
117     OutputLayer[i].Target = OutputLayer[i].Value.CompareTo(output) == 0 ?
          1.0 : 0.0 ;
118     OutputLayer[i].Error = (OutputLayer[i].Target - OutputLayer[i].output) *
          (OutputLayer[i].output) * (1 - OutputLayer[i].output) ;
119   }
120 }
```

Figure 7: Code pour la fonction "ForwardPropagation"

attribués aléatoirement. Ensuite, chaque élément de la bibliothèque est placé en entrée et reconnu par le système, un par un. A la sortie se retrouve ce que le système a déduit en fonction du poids de la connexion le plus élevé. Selon ce que le système a reconnu en sortie, l'utilisateur pourra vérifier s'il y a eu une erreur, par rapport aux données mises en entrée. En cas d'erreur, le poids de la connexion est diminué, sinon augmenté. Ce cycle se reproduira en boucle jusqu'à atteindre au final une erreur proche du millième, très faible. Pour finir, un fichier sérialisé est créé et reprend les paramètres comme le nombre de couches, le poids de chaque liaison du système, etc. Ce fichier est chargé pour la reconnaissance, permettant de rendre l'entrainement unique pour un afficheur.

```
121 //Fonction de propagation en arrière des données pour l'entrainement
122 public void BackPropagate()
123 {
124   //Pour le nombre d'éléments contenus dans la sortie
125   for (int j = 0 ; j < OutputNum ; j++)
126   {
127     //Pour le nombre d'éléments contenus dans l'entrée
128     for (int i = 0 ; i < PreInputNum ; i++)
129     {
130       //On modifie les poids des liens de la couche d'entrée selon l'erreur
              de sortie avec un coefficient
131       PreInputLayer[i].Weights[j] += LearningRate * (OutputLayer[j].Error) *
              PreInputLayer[i].Value ;
132     }
133   }
134 }
```

Figure 8: Code pour la fonction "BackwardPropagation"

Pour finir, lorsque le système est prêt, la reconnaissance peut être effectuée (figure 10). Pour ce faire, il suffit de placer le caractère en entrée, après traitement de l'image, comme expliqué plus haut, le caractère lié avec le plus grand poids est mis en sortie. Le système aura reconnu au mieux le caractère mis en entrée.

La grande force de ce système est qu'il est capable de déduire, grâce à l'entrainement préalable, la meilleure sortie possible. Si l'image du caractère est trouble, mal cadrée, de couleur différente, de luminosité différente, etc, le système sera toujours capable de reconnaitre un caractère, mais peut-être pas le bon.

```
135 //Fonction permettant l'entraînement
136 public bool Train()
137 {
138   //Déclaration de variables
139   double currentError = 0 ;
140   int currentIteration = 0 ;
141   //Construction dun nouvel argument de réseau
142   NeuralEventArgs Args = new NeuralEventArgs() ;
143   do
144   {
145     //On initialise la variable pour l'erreur actuelle
146     currentError = 0 ;
147     //Pour chaque valeur de clé du set Entrainement
148     foreach (KeyValuePair<T, double[]> p in SetEntrainement)
149     {
150       //On va faire appel aux fonctions de propagation du réseau
151       //d'abord en Forward
152       NeuralNet.ForwardPropagate(p.Value, p.Key) ;
153       //Puis en Backward
154       NeuralNet.BackPropagate() ;
155       //On calcule l'erreur
156       currentError += NeuralNet.GetError() ;
157     }
158     //On incrémente la variable pour le nombre de passages du système
159     currentIteration++ ;
160     //Si la valeur de l'erreur a changé et qu'il ne tourne plus
161     if (IterationChanged != null && currentIteration % 5 == 0)
162     {
163       //On renvoie les valeurs du nombre de passages et de l'erreur
164       Args.CurrentError = currentError ;
165       Args.CurrentIteration = currentIteration ;
166       IterationChanged(this, Args);
167     }
168   }
169   //On l'exécute jusqu'à une erreur en dessous de 0.001 et un nombre de
          passages plus petit que 100 000
170   while (currentError > 0.001 && currentIteration < maximumIteration &&
          !Args.Stop) ;
171   //Si changement dans le nombre de passages
172   if (IterationChanged != null)
173   {
174     //On renvoie les valeurs du nombre de passages et de l'erreur
175     Args.CurrentError = currentError ;
176     Args.CurrentIteration = currentIteration ;
177     IterationChanged(this, Args);
178   }
179   //Si on a dépassé le nombre maximum d'itérations, on retourne la fonction
          fausse
180   if (currentIteration >= maximumIteration || Args.Stop)
181   return false ;
182   return true
183 }
```

Figure 9: Code utilisé pour l'entrainement

4. Mise en place du projet

Le projet va réutiliser chaque partie expliquée ci-dessus pour l'acquisition et la reconnaissance. Le programme va d'abord être divisé en 5 modules : le module "Main", où l'utilisateur va pouvoir rentrer ses paramètres, le module "Entrainement", pour exécuter l'entrainement, le module "Acquisition et reconnaissance" qui fera l'acquisition et la reconnaissance (ce module restera en arrière plan, non visible par l'utilisateur), le module "Affichage" qui permettra à l'utilisateur de contrôler le système grâce à un graphique et aux tableaux des valeurs reconnues, et pour finir un module de traitement de données, pour vérifier les valeurs reconnues ainsi que pour récupérer les dix valeurs les plus stables pour l'étalonnage.

```
184 //Fonction de reconnaissance
185 public void Reconnaissance(double[] Input, ref T MatchedHigh, ref double
        OutputValueHight,ref T MatchedLow, ref double OutputValueLow)
186 {
187   //Déclaration d'entiers
188   int i, j;
189   //Déclaration de doubles
190   double max = -1;
191   double total = 0.0;
192   //On applique les entrées au réseau
193   for (i = 0; i < PreInputNum; i++)
194   {
195     //On place les valeurs d'entrée dans la couche d'entrée
196     PreInputLayer[i].Value = Input[i];
197   }
198   //Pour tous les éléments de sortie
199   for (i = 0; i < OutputNum; i++)
200   {
201     //On initialise la variable total
202     total = 0.0;
203     //Pour tous les éléments de la couche d'entrée
204     for (j = 0; j < PreInputNum; j++)
205     {
206       //On calcule chaque valeur d'entrée multipliée par le poids
              correspondant
207       total += PreInputLayer[j].Value * PreInputLayer[j].Weights[i];
208     }
209     //On place la valeur dans la sortie
210     OutputLayer[i].InputSum = total;
211     //On place en sortie le total traité par la fonction d'activation
212     OutputLayer[i].output = (1 / (1 + Math.Exp(-total))) ;
213   }
214 }
```

Figure 10: Code utilisé pour la reconnaissance

L'utilisation des modules permet une grande flexibilité, chaque module étant complètement indépendant et pouvant être lancé plusieurs fois. De plus, en cas de mise à jour ou de modification d'un module, les autres modules ne sont pas concernés et peuvent continuer de fonctionner. Ceci permet que si une des quatre caméras se déconnecte, de ne pas affecter l'acquisition des autres en bloquant et arrêtant le programme.

La communication entre les modules se fait par passage d'arguments lorsqu'on ouvre les modules, comme le numéro de la caméra, le numéro de l'appareil, etc. Des fichiers de type texte seront également utilisés à chaque étape du programme comme sauvegarde des valeurs. Les fichiers texte sont très faciles à utiliser ou à lire et écrire, même sans le logiciel, avec un simple bloc note. Le principe général du système avec module est présenté à la figure 11.

Le module servant à la gestion des données utilise une moyenne flottante avec un calcul de l'écart type afin de déterminer le palier de stabilité de la mesure. Pour cela, il recherche pour la caméra 1 (par défaut l'étalon de mesure) trente mesures stables, c'est à dire ne variant pas au millième. Ensuite, le programme enregistre le numéro de l'acquisition et va chercher simultanément dans les trois autres acquisitions si dix mesures sont stables respectivement au dernier chiffre significatif. Si tel est le cas, ces dix mesures sont prises en même temps pour chacune. Sinon, la liste des valeurs est descendue trente mesures plus loin pour l'étalon. Si ce n'est toujours pas le cas, la valeur stable à l'avant dernier chiffre significatif est prise. Ce schéma est suivi indéfiniment jusqu'à déterminer la meilleure stabilité simultanée pour toutes les acquisitions.

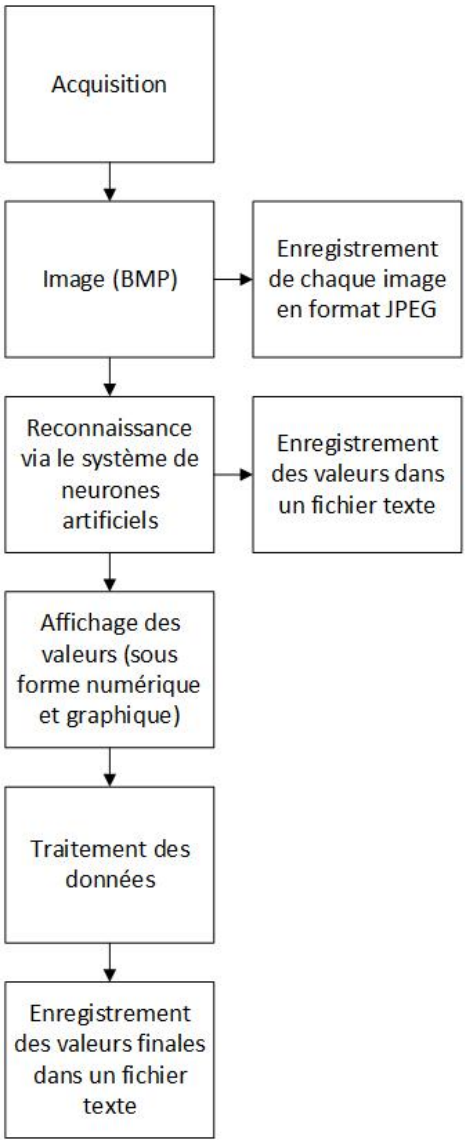

Figure 11: Séquence du système général avec modules

5. Conclusions

De nombreux tests ont été effectués, en préétude avant de commencer le projet, mais également à la fin de celui-ci.

Des tests avec des périodes allant de deux à soixante secondes sur une durée de cinq minutes à dix heures ont été effectués et démontrent la stabilité ainsi que les limites du systèmes. Effectivement, en dessous de deux secondes, le système subit des ralentissements.

Au niveau des images prises, de nombreux types d'afficheurs (exemple dans la section 7.) ont été testés, de type 7-segments et pixel, de toutes couleurs, rétro luminescent ou non. Il a même été démontré que le système est capable de reconnaitre des mesures sur des afficheurs inclinés à 45 degrés, ou lorsque l'image est rognée par rapport à celle de la bibliothèque. Le fait de ne travailler qu'avec dix chiffres à reconnaitre rend le système très stable et efficace.

Enfin des derniers tests ont été effectué en cas réel, c'est à dire comme le technicien sera à même d'employer ce système, avec les appareils provenant de clients à étalonner. Ces tests finaux ont été très concluants, montrant un système fonctionnel en pratique.

En conclusion, le projet a bien été mené à son terme, répondant correctement au cahier des charges précisés et au but d'automatisation fixé. Ce projet apporte également un regard nouveau sur les nouvelles technologies, tel que le système neuronal, qui est amené à être exploité considérablement dans le futur dans de nombreux domaines.

6. Remerciements

Je tiens à remercier M. Roland Platteau, administrateur délégué et responsable technique chez LBT Testing & Calibration, ainsi que tous ses employés pour toute l'aide et le suivi de ce projet.

Je tiens également à remercier Laura Poffé pour son soutien et ses bonnes relectures.

7. Illustrations

Sources

[1] ADMASSU, D. (consulté le 8 septembre 2013), *Unicode Optical Character Recognition*.
Adresse URL : http://www.codeproject.com/articles/15304/unicode-optical-character-recognition.

[2] CHERKASOV, A. (consulté le 8 septembre 2013), *Creating Optical Character Recognition (OCR) applications using Neural Networks*.
Adresse URL : http://www.codeproject.com/Articles/3907/Creating-Optical-Character-Recognition-OCR-applica.

[3] CODEPROJECT, (consulté le 8 septembre 2013), *Acceuil*.
Adresse URL : http://www.codeproject.com/.

[4] FIRAT, M. (consulté le 8 septembre 2013), *Image Recognition with Neural Networks*.
Adresse URL : http://www.codeproject.com/Articles/19323/Image-Recognition-with-Neural-Networks.

[5] GOOGLE, (consulté le 8 septembre 2013), *Tesseract-ocr*.
Adresse URL : https://code.google.com/p/tesseract-ocr/.

[6] MAGOMEDOV, B. (consulté le 8 septembre 2013), *Hopfield model of neural network for pattern recognition*.
Adresse URL : http://www.codeproject.com/Articles/15949/Hopfield-model-of-neural-network-for-pattern-recog.

[7] ZAVIO, (consulté le 8 septembre 2013), *Zavio F1105*.
Adresse URL : http://www.zavio.com/product.php?id=58.