

RAPPORT DE STAGE - TN09 - GI

Développement de l'interface homme-machine pour l'inspection automatisée

13 FÉVRIER 2023 - 28 JUILLET 2023

Auteur :
Zhentao XU

Encadrant UTC :
Véronique CHERFAOUI

Encadrant entreprise :
Julien MONVILLE

Entreprise :
Deltacad
795 rue de Longues Rayes
60610, Lacroix Saint-Ouen

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé lors de la rédaction de ce rapport.

Tout d'abord, je tiens à remercier vivement Vincent ARGENTON, mon maitre de stage, pour le temps qu'ils m'a toujours accordé sans hésitation, ainsi que pour les opportunités et responsabilités qu'il m'a offertes.

Ensuite, je voudrais remercier le manager de l'équipe, Stéphane GLOAGUEN, pour la reconnaissance, le temps et les conseils qu'il m'a accordé.

De plus, je souhaite aussi remercier Kevin HELOUART, qui, avec Vincent, a su m'intégrer à l'équipe très rapidement.

Enfin, je remercie Michelangelo NERI et Milan RADOVIC d'avoir accepté ma candidature.

Je souhaite aussi remercier tout mes collègues pour le temps passé à leurs côtés, ainsi que pour leurs précieux conseils, et plus spécialement Thibault, Bastien et Dominique.

Résumé technique

Mon stage se déroule à Deltacad, une société d'ingénierie spécialisée en informatique scientifique. En collaboration avec une entreprise AML System et deux instituts de recherche, les laboratoires Roberval et Heudiasyc de l'UTC, nous développons un logiciel pour des scénarios industriels, qui entraîne des modèles d'apprentissage automatique à partir d'images de pièces sur la chaîne de production et les applique pour déterminer la qualité des pièces et détecter les défauts.

Dans ce cadre, ma tâche consistait à concevoir et à développer l'interface homme-machine et à améliorer progressivement sa fonctionnalité en fonction des besoins. Durant mon stage, J'ai réalisé une interface du logiciel extensible en utilisant WPF et j'ai conçu l'architecture globale du logiciel, en intégrant des modules fonctionnels déjà conçus ainsi que certains de mes propres modules fonctionnels. Par ailleurs, j'ai également utilisé certaines extensions externes, telles que l'affichage des journaux d'opérations via le cadre POA (programmation orientée aspect), et l'enregistrement et l'importation des informations de configuration via la sérialisation xml.

Mots-clés : .Net WPF, MVVM(Model-View-ViewModel), C#, POA, python, C++

Introduction

Dans le cadre de ma formation d'ingénieur à l'Université de Technologie de Compiègne, le troisième semestre est consacré à un stage de 24 semaines d'assistant ingénieur. Cette expérience comme une longue période de travail professionnel, nous permet d'avoir un aperçu du fonctionnement de l'entreprise et du monde professionnel, d'appliquer les connaissances acquises lors de sa formation, d'exprimer notre créativité et nos idées jeunes et énergiques. Et dans ce rapport de stage, je résume le travail que j'ai effectué à Deltacad au cours des six mois allant de février 2023 à juillet 2023 et je le présente plus en détail en trois partis principaux.

Dans la première partie, je vous donnerai une vue d'ensemble de mon entreprise et mon point de vue sur mon équipe. Cela vous donnera un premier aperçu de l'environnement de mon stage.

Dans la deuxième partie, je vous présenterai quelques détails sur l'ensemble de mon stage, y compris le sujet du stage, l'organisation des tâches, ma contribution et les aspects techniques du projet. En résumé, cette section est plus orientée vers une vision globale et n'est pas divisée en modules individuels.

Dans la troisième section, je décomposerai mes réalisations au cours de mon stage en un certain nombre de points afin de les expliquer plus en détail. Je également expliquerai les raisons pour lesquelles j'ai choisi certaines options, ainsi que les difficultés et solutions techniques rencontrées à ce moment-là. Dans cette partie, je me concentrerai sur les aspects techniques et, en raison de la spécificité du projet, je parlerai des différents modules du projet pour vous donner une compréhension plus claire.

Enfin, je résumerai les améliorations que j'ai apportées pendant mon stage et je donnerai quelques indications sur le développement futur du projet.

1. Présentation de Deltacad

1.1 Le groupe Deltacad

1.1.1 Historique du groupe

2. Mission

2.1 Sujet de stage

Le sujet de mon stage était initialement Développement/Intégration de fonctions de Vision par ordinateur pour l'inspection automatisée. Durant toute la période de mon stage, mon travail s'est axé autour d'un logiciel appelé **PowerEye**. **PowerEye** est un logiciel développé pour des scénarios industriels qui utilise la vision par ordinateur pour détecter les défauts des pièces sur la chaîne de production. Les laboratoires de l'UTC faisant partie de ce projet, ils assurent le développement des machines apprenantes. Deltacad est responsable de la conception de l'interface de l'ensemble du projet et de l'intégration des autres fonctions.

Avant que je ne commence mon stage, **PowerEye** avait déjà été conçu avec certaines fonctionnalités ainsi qu'un prototype d'interface. Cependant, il était clair que le système n'était pas encore adapté à une application formelle au sein de l'entreprise. Par conséquent, la tâche principale de mon stage consistait à concevoir et améliorer une interface homme-machine plus pratique basée sur le prototype de l'interface et essayer de tester les pièces d'AML Systems¹ à l'aide de cette interface après intégration de la fonction.

En ce qui concerne les lignes directrices spécifiques pour la conception de l'interface, le chef de projet et le suiveur m'ont fourni un aperçu général des différents modules qu'elle contiendrait et des fonctionnalités qu'elle pourrait inclure. Après avoir confirmé le thème et le programme de travail, mon stage a officiellement commencé.

1. une société du segment éclairage du groupe Johnson Electric, conçoit, produit et commercialise, des solutions pour améliorer la visibilité, la sécurité, et le confort du conducteur

2.2 Planning de stage

Au début de mon stage, mes tâches étaient planifiées en trois phases :

- **1.** Développement de l'interface homme-machine de PowerEye
- **2.** Test sur les données de l'entreprise partenaire AML système
- **3.** Déploiement d'applications dans le système AML

Dans le cadre du travail réel, pour chaque phase de la tâche, je l'ai décomposée en différentes étapes. La première phase peut en fait être divisée en plusieurs parties : la conception de l'interface, la réalisation de l'apparence de l'interface, la réalisation de la structure des données de l'interface et la réalisation des fonctions de la couche d'application de l'interface. La deuxième phase comprend également l'intégration de la fonctionnalité conçue, le test des données et l'amélioration de la fonctionnalité. Il est très regrettable qu'en raison de contraintes de temps, mon stage se soit terminé à la deuxième phrase. Mais en fait, en plus des tâches incluses dans le plan, j'ai également complété un certain nombre d'autres fonctionnalités, y compris enregistrement des appels de fonction et exécution de programmes de script `c#` via des chaînes de caractères

À mon avis, la partie la plus importante du processus de stage est la structure globale du projet, car elle influe grandement sur le développement des idées et l'évolutivité du projet. À cette fin, j'ai amélioré l'architecture globale à plusieurs reprises au cours du processus de développement, en utilisant des `prism` au début de projet individuel, en divisant l'interface en couches d'affichage et d'application, et en développant avec le cadre `mvvm`.

2.3 Contributions

Comme mentionné dans la section précédente, lorsque j’ai commencé à travailler sur le projet, il existait déjà un prototype d’interface ainsi que certains modules fonctionnels.

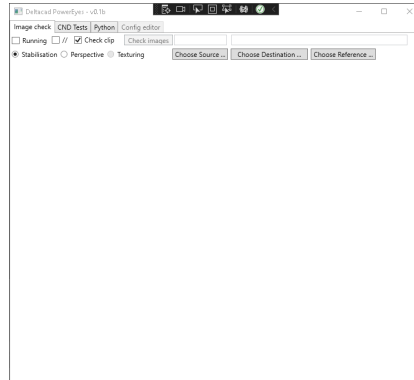


FIGURE 1 – Prototype d’interface PowerEye

L’interface se compose de quatre modules : Image check, CND Tests, python et config editor. Il contient des méthodes d’application de Template Matching pour la détection des pièces et des modèles d’entraînement pour les tests avec des ensembles de données. Mais le fait est que l’interface prototype n’est pas suffisante pour inclure toutes les fonctionnalités envisagées par PowerEye. Par conséquent, ma principale contribution tout au long du stage a été la conception et le développement d’une nouvelle version de l’interface PowerEye. À la fin du stage, la nouvelle version de l’interface de powereye ressemblait à ceci.

Il contient cinq modules :

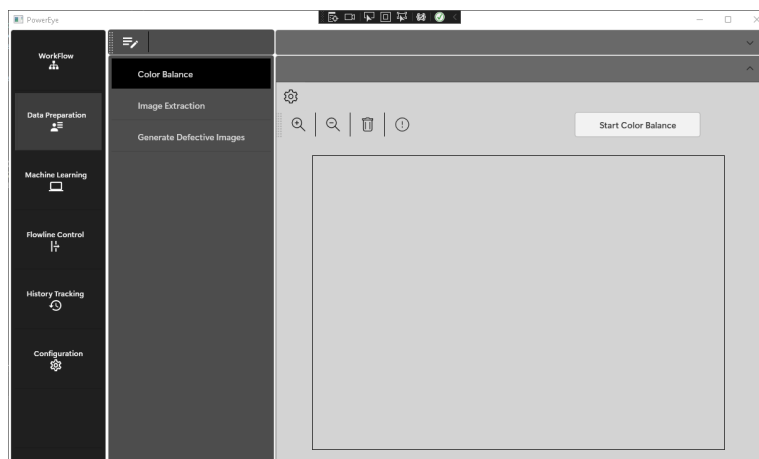


FIGURE 2 – Interface PowerEye redessinée

- **Préparation des données (Data preparation)** : Ce module permet d’extraire les contours des pièces de l’image et d’ajouter du bruit de perlin à l’image.

- **Apprentissage automatique (machine learning)** : Ce module est utilisé pour entraîner le modèle ckpt à partir de l'ensemble de données.
- **Contrôle de la chaîne d'assemblage (flowline control)** : Ce module détecte si une pièce est qualifiée par Template Matching, et enregistre les données du résultat du test.
- **Interroger les données historiques (History tracking)** : Ce module interroge les données du fichier Excel par mots-clés.
- **Configuration des données globales (configuration)** : Ce module affiche la configuration complète du logiciel en cours et l'importe par l'intermédiaire de fichiers de configuration.

En plus de la partie principale du logiciel, j'ai mis en place les fonctionnalités suivantes

- Utilisation du cadre AOP pour afficher les actions de la couche applicative.
- Implémentation d'un script c# qui s'exécute à partir d'une chaîne de caractères.
- Implémentation d'ajouts manuels de fonctionnalités existantes pour personnaliser le flux de travail.

Ce qui ci-dessus résume essentiellement toutes les contributions que j'ai apportées au cours de mon stage. Dans mon travail, j'ai conçu et développé le projet de manière indépendante en utilisant les exigences comme point de départ, et j'ai finalement bien accompli la tâche.

2.4 Outils et technologies

Tout au long de mon stage, j'ai travaillé avec les outils suivants :

— **Pour le développement :**

— *Comme langage de programmation* : C#, WPF.NET, C++, Python

— *Comme environnement de développement* : Microsoft Visual Studio, **IDE** complet pour les développeurs .NET et C++ sur Windows.

— **Pour la bureautique** : **LaTeX**, pour la rédaction de ce rapport. WebEx, pour les réunions de projet à distance

— **Pour la gestion des différentes versions du logiciel** : **SmartSVN**, un système de gestion des versions.

Pour la réalisation du projet, j'ai utilisé les extensions suivantes :

— **WPF-UI** : Extension pour modifier le style du contrôle

— **MahApps.Iconpacks** : Extension pour l'ajout d'icônes de boutons

— **Fody** : Extension pour la mise en œuvre du cadre aop

— **YamlDotNet** : Extension pour les modifications de l'édition yaml

2.5 Validation de travaux

La validation de mes travaux se déroulait lors de réunions avec l'équipe de projet. Comme le chef de projet ne travaille pas au même endroit que moi, nos réunions se passaient généralement à distance via le WebEx.

Au cours de la réunion, normalement, je commençais par présenter et démontrer l'état d'avancement de la tâche. Ensuite, le tuteur et le chef de projet discutaient avec moi des améliorations à apporter et planifient les tâches suivantes. le contenu de chaque réunion était enregistré par le chef de projet et m'était envoyé par e-mail après la réunion.

Cela me permet de partager mon point de vue sur le projet et me donne une idée claire de ce qui doit être fait a la phase actuel, ce qui améliore grandement mon efficacité et standardise mon travail.

2.6 Prise de recul

2.6.1 Intérêt de mon travail

L'intérêt de mon travail pour l'équipe était de concevoir et développer une IHM plus aboutie que la précédente, migrer et améliorer les fonctionnalités développées précédemment. En outre, le cadre aop et le modèle de l'observateur sont plus propices à la maintenance de l'interface, et il existe un bon cadre général pour le développement du suivi.

2.6.2 Améliorations à envisager

Mais tout comme le développement de nombreux logiciels connus nécessite beaucoup de polissage, le logiciel que j'ai développés au cours de mon stage de six mois n'étaient certainement pas parfait. Dans le développement futur du projet, les améliorations suivantes sont nécessaires.

- Développer la fonctionnalité de certains modules et les intégrer dans l'interface.
- Extension de l'utilité de l'interface, par exemple par l'ajout de la touche de raccourci.
- Améliorer les fonctionnalités pour s'adapter aux différents produits de l'industrie.

2.6.3 Réflexions sur l'impact de mon travail

Selon moi, ma contribution permettra aux utilisateurs de simplifier le processus, de réduire le coût d'apprentissage de l'interface et de rendre la visualisation des résultats plus intuitive.

3. Réalisations

Dans la section précédente, j’ai déjà mentionné que mon stage était principalement axé sur **PowerEye**. Dans cette section, je détaillerai mes réalisations, en commençant par l’architecture globale, puis les modules individuels, et enfin certaines fonctionnalités en dehors du corps principal du logiciel.

3.1 Architecture globale de PowerEye

Dans cette partie, je présenterai la composition du **PowerEye** en termes d’architecture global, y compris le choix de modèle de conception, la construction de la structure de données de l’interface et l’apparence de l’interface. De cette façon, vous pouvez d’abord avoir un concept général du **PowerEye**, et comprendre plus facilement la conception du chaque module dans les parties suivant.

3.1.1 Modèle de conception

Le choix du modèle de conception influe grandement sur la manière de développer le programme. Au début du développement du **PowerEye**, j’ai choisi MVVM comme modèle de conception.

Présentation du MVVM

Il existe trois composants principaux dans le modèle MVVM : le modèle, la vue et le modèle de vue. Chacun sert un objectif distinct. La vue est chargée de définir la structure, la disposition et l’apparence de ce que l’utilisateur voit à l’écran. Le modèle de vue implémente des propriétés et des commandes avec lesquelles la vue peut effectuer une liaison aux données. Les classes de modèle sont des classes non visuelles qui encapsulent les données de l’application. Ainsi, le modèle peut être considéré comme une représentation du modèle de domaine de l’application, qui comprend généralement un modèle de données avec une logique métier et une logique de validation.

Application pratique

Au début de la conception du **PowerEye**, nous avons décidé d'utiliser **WPF** pour le développement du logiciel. Et **WPF** offre une fonctionnalité très importante, à savoir la liaison de données. C'est le processus qui établit une connexion entre l'IU du logiciel et les données affichées. Si la liaison est correctement paramétrée et si les données fournissent les notifications appropriées, lorsque les données changent de valeur, les éléments qui sont liés aux données reflètent automatiquement ces changements. Comme le montre la figure ci-dessous, lorsque la source de la liaison de données est modifiée, la valeur target de la liaison est modifiée en conséquence. En pratique, nous pouvons lier les propriétés des contrôles au modèle des données.

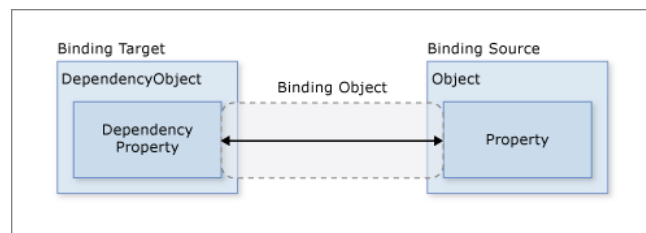


FIGURE 3 – La liaison de données

Grâce à cette caractéristique, il est facile d'utiliser les changements apportés à la partie modèle de MVVM pour conduire des changements dans l'affichage de l'interface. Plus précisément, lors du développement du **PowerEye**, j'utilise des informations de données telles que les chemins sélectionnés et les fichiers ouverts comme la partie modèle. Et le modèle de vue contient la partie modèle et les fonctions de contrôle des événements. Enfin, l'apparence de l'interface apparaît comme la vue. En outre, pour rendre la structure globale plus claire, j'ai établi une distinction entre la couche de présentation et la couche d'applications.

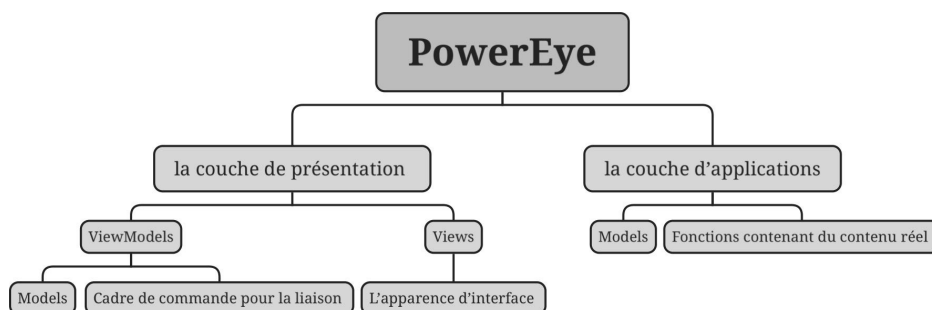


FIGURE 4 – La structure globale du PowerEye

Avantages de ce choix

L'utilisation de ce modèle de conception facilite grandement l'ensemble du processus de développement. Premièrement, il sépare le développement de l'apparence de l'interface de l'ensemble. Une fois l'apparence de l'interface développée, l'adaptation des données peut être réalisée en modifiant simplement la source de la liaison de données. De plus, nous pouvons réutiliser le modèle de vue pour des interfaces similaires. Enfin, comme la couche d'application est séparée, il est possible d'appeler directement des fonctions qu'elle contient pour tester.

3.1.2 Structure des données

Les données font partie du modèle dans le cadre de MVVM, je les ai également conçu pour répondre aux besoins. Pour chaque module, j'ai créé une sous-classe avec une liste pour stocker les instances qui ont été créées. Afin que les différentes interfaces d'un même module puissent être enregistrées dans la même liste, j'ai créé une classe de base. Enfin, les classes de chaque interface héritent simplement de la classe de base et modifiée différemment en fonction des besoins. En outre, la classe de base doit hériter de la classe `BindingBase` pour implémenter la fonctionnalité de notification de l'interface lorsqu'une propriété est changée. Et voici un aperçu de la structure des données.

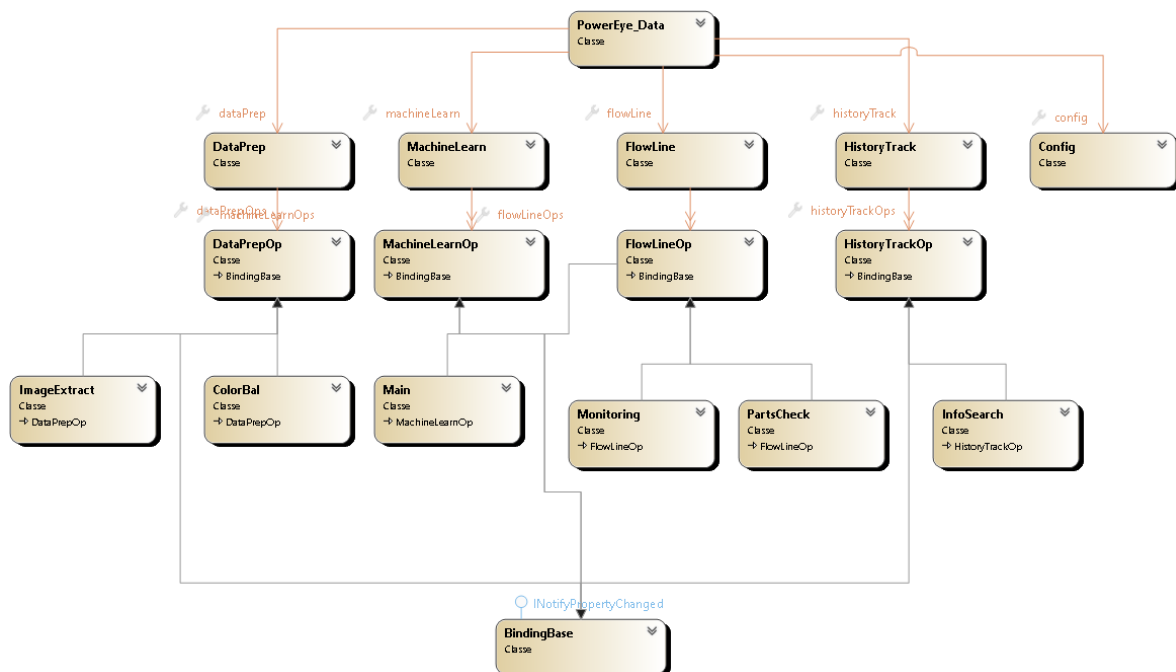


FIGURE 5 – La structure des données du PowerEye

3.1.3 Disposition de l'interface graphique

Après avoir présenté la partie du modèle, j'aimerais expliquer la conception de la disposition de l'interface graphique, c'est-à-dire la partie concernant la vue dans le cadre de MVVM. La FIGURE 2 montre le style d'interface du **PowerEye**, qui semble se composer de trois parties. Mais ce n'est pas vraiment ce que j'ai fait avec l'interface au début. Au départ, j'ai choisi le **TabControl** pour séparer les modules et les développer à partir de là. L'interface conçue à ce moment-là est illustrée ci-dessous.

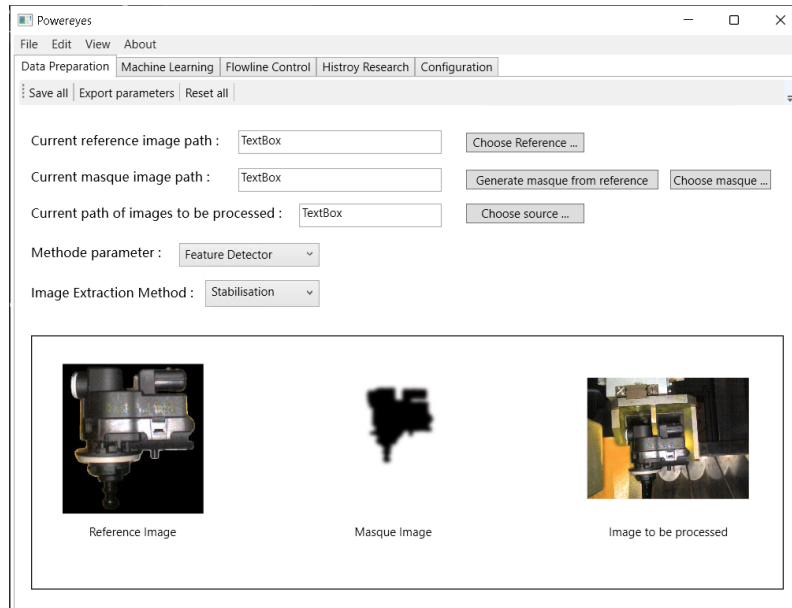


FIGURE 6 – La Conception initiale de l'interface

Cependant, le problème est double : d'une part, chaque module n'a pas nécessairement une seule fonction, et les combiner dans une interface unique aurait l'air encombré. D'autre part, avec cette approche, le développement de toutes les interfaces des modules se fait dans le même fichier, ce qui réduit la lisibilité du code.

J'ai donc finalement choisi d'abandonner cette solution et d'utiliser **Contentcontrol** pour la disposition de l'interface. Plus précisément, j'ai divisé l'interface globale en trois parties : la barre des modules, la barre des fonctions et la fenêtre principale. En sélectionnant le module, différentes barres de fonctions s'affichent, puis la fenêtre principale est affichée en cas de sélection de la fonction. Et grâce à la caractéristique du **Contentcontrol**, nous pouvons injecter l'interface en tant que contrôle, ce qui permet le développement indépendant des différentes parties. La disposition finale de l'interface est illustrée ci-dessous.

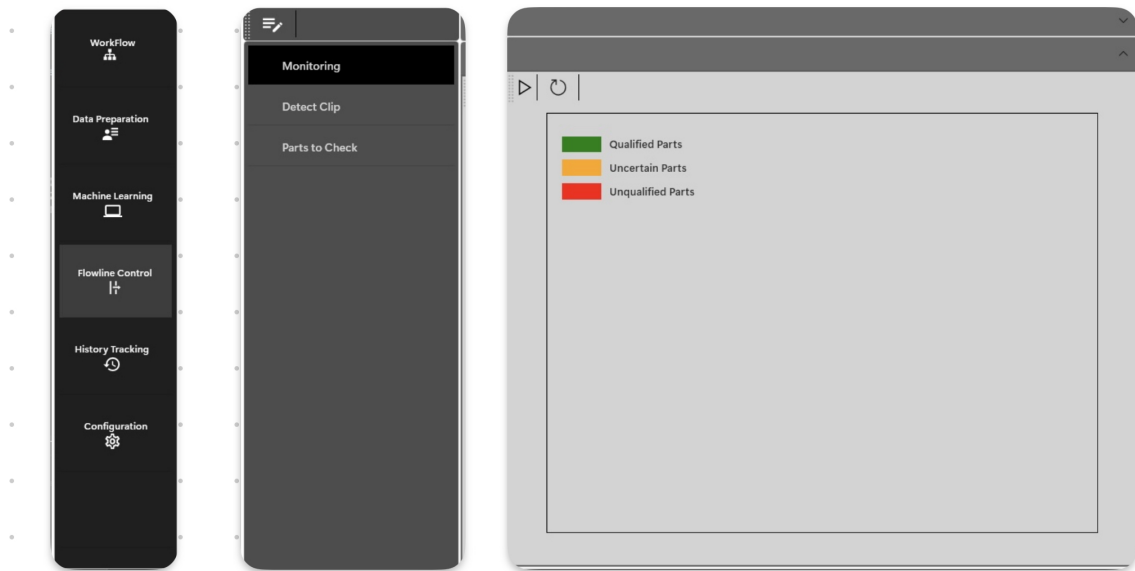


FIGURE 7 – La disposition finale de l’interface

Dans la partie de la fenêtre principale, compte tenu du fait que la réalisation de nombreuses fonctions nécessite la sélection du chemin d’accès au fichier et d’autres configurations, j’ai également fait la conception correspondante. L’**Expander** est utilisé pour diviser la fenêtre en deux parties, la partie supérieure pour les choix de configuration et la partie inférieure pour les fonctionnalités. La figure suivant montre spécifiquement la fenêtre principale.

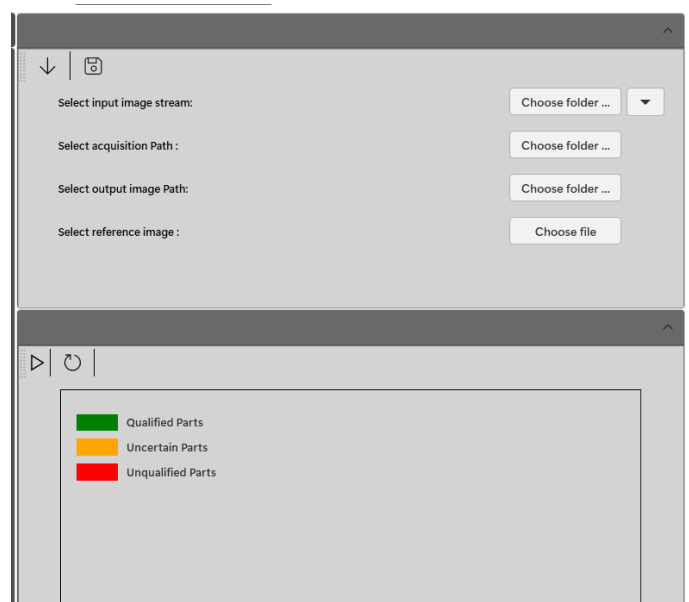


FIGURE 8 – la fenêtre principale

3.2 Multiples modules pour PowerEye

Après l'introduction générale du PowerEye, cette section va le présenter en détail à partir de chaque module.

3.2.1 Préparation des données

Le traitement des données étant une condition préalable à la formation au modèle, un module a également été spécialement conçu dans PowerEye pour mettre en œuvre les fonctions correspondantes. Dans ce module, nous avons actuellement développé deux interfaces entièrement fonctionnelles, à savoir l'extraction d'images et l'ajout du bruit de perlin.

Extraction d'images

L'objectif de cette interface est d'extraire le contour de la pièce sur le photo de la ligne d'assemblage. Lorsque j'ai commencé à développer PowerEye, une fonction qui utilise le **feature matching** pour réaliser cet objectif a déjà été développée. Donc ma tâche principale a consisté à l'intégrer dans l'interface et à y apporter des améliorations.

Lors de la conception de l'interface graphique pour cette fonctionnalité, j'ai suivi l'idée générale selon laquelle la partie supérieure de l'interface est utilisée pour la sélection de la configuration. Dans la partie inférieure de l'interface, j'ai conçu une barre de progression et une fenêtre pour montrer l'image avant et après le traitement.

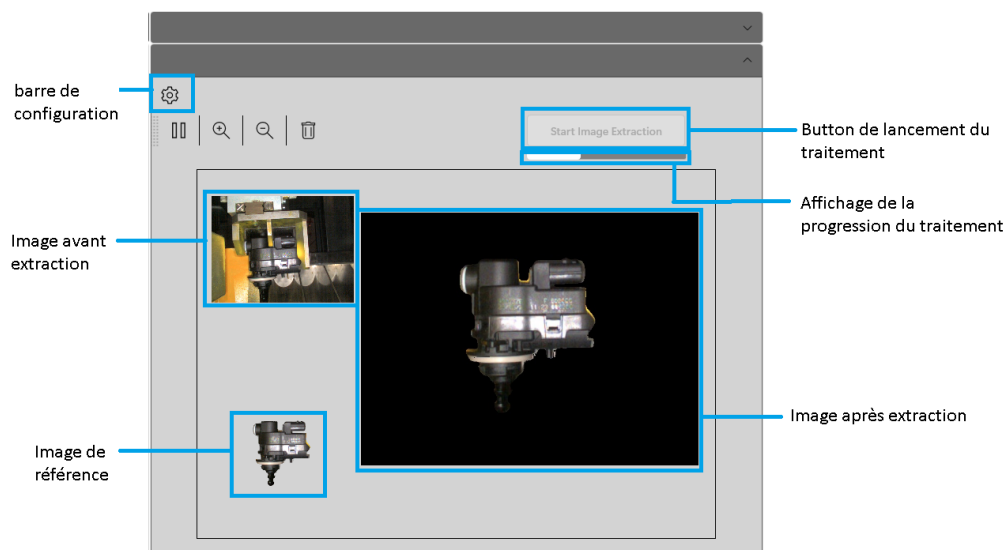


FIGURE 9 – l'interface graphique d'extraction d'images

En ce qui concerne l'intégration de la fonctionnalité d'extraction développée, j'ai résolu deux problèmes principaux. D'une part, la fonction est développée en C++, tandis que l'interface est développée en C#, il faut donc choisir la bonne méthode pour l'appeler. En final, j'ai choisi d'utiliser la même méthode que l'interface prototype, en appliquant le fichier `dll` pour l'importation et l'exportation des fonctions. D'autre part, c'est un problème très courant dans le développement d'interface graphiques est de savoir comment exécuter la fonction sans bloquer l'interface graphique. Donc le `thread` est utilisé pour résoudre ce problème. J'ai intégré la fonction extraite dans un `thread` et appliqué `FileWatcher` pour détecter les images ajoutée au répertoire afin de les afficher. Plus précisément, il fonctionne de la manière suivante :

1. Après de cliquer sur le bouton de démarrage, un nouveau `thread` est créé pour exécuter la fonction d'extraction d'image.
2. `FileWatcher` surveille toujours le répertoire de destination et détecte si les images traitées ajoutées sont occupées.
3. Lorsque l'image n'est plus occupée, modifie le propriété de la source de liaison dans le `thread` principal pour l'afficher.

Ajout du bruit de perlin

L'objectif de cette interface est d'ajouter un masque de bruit à l'image pour générer des pièces comportant des défauts. Comme dans la partie précédent, il existe déjà une fonction permettant d'ajouter des masques de bruit, développée à l'aide de python. L'idée de conception et la manière de fonctionner de cette partie sont similaires à la précédente, que je vais brièvement présenter ici. L'interface graphique est présentée comme ci-dessous.

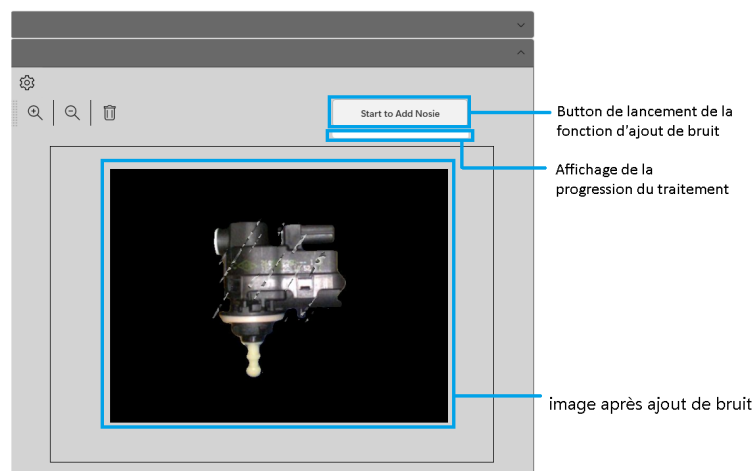


FIGURE 10 – l'interface graphique d'ajout du bruit

Et dans la section, je vais me concentrer sur la méthode d'appel python et sur une amélioration que j'ai réalisée. Pour l'appel au module python, je me réfère à l'implémentation de l'interface prototype. La classe `process` en C# me permet d'exécuter python et de passer le script à lancer comme les paramètres de démarrage. Et dans la mise en œuvre, j'ai également encapsulé le processus python, y compris les valeurs d'entrée et de sortie, dans une classe pour le réutiliser dans le module d'apprentissage automatique.

En termes d'améliorations, la fonction d'ajout de bruit nécessite trois paramètres d'entrée : une image de la pièce, un masque de bruit et un masque de forme.

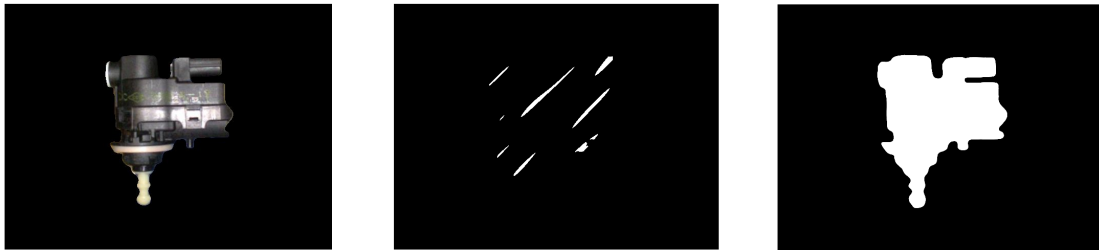


FIGURE 11 – trois paramètres d'entrée

Mais en effet, il n'est pas difficile d'obtenir le masque de forme d'une pièce à partir de l'image initial. Par conséquent, j'ai développé la fonction pour générer les masques de forme afin de réduire la paramètre d'entrée. Les étapes pour réaliser cette fonction sont les suivantes :

1. Chargement et conversion de l'image d'entrée en niveaux de gris et Application d'un flou gaussien.
2. Calcul de la magnitude et de la direction du gradient à l'aide de l'algorithme de Scharr.
3. Normalisation de la magnitude du gradient dans la plage $[0, 255]$.
4. Opération de fermeture morphologique et recherche du plus grand contour d'arête.
5. Générer et exporter des masques transparents avec des contours de remplissage maximaux.

Après avoir terminé cette partie de l'amélioration, le module de préparation des données est désormais plus robuste, ce qui permet de passer au module d'apprentissage automatique qui utilise les données.

3.2.2 Apprentissage automatique

3.2.3 Contrôle de la chaîne d'assemblage

Ce module, qui est l'application la plus importante de tout le logiciel, permet de détecter les images des pièces sur la ligne d'assemblage et de sauvegarder les données.

Surveillance de la ligne d'assemblage

L'idée de cette fonction est d'utiliser le modèle entraîné pour inspecter chaque image de pièce et d'afficher les résultats de l'inspection dans l'interface. Mais pendant le développement, il était impossible d'accéder aux capteurs dans la scène réelle. J'ai donc utilisé un processus qui copie les images dans un répertoire toutes les cinq secondes comme une alternative.

Dans la conception de l'interface, j'ai choisi d'utiliser différentes couleurs pour indiquer si la pièce est qualifiée ou non, un peu comme le voyant d'avertissement, et afficher l'image de ce pièce. L'interface graphique conçue est illustrée ci-dessous.

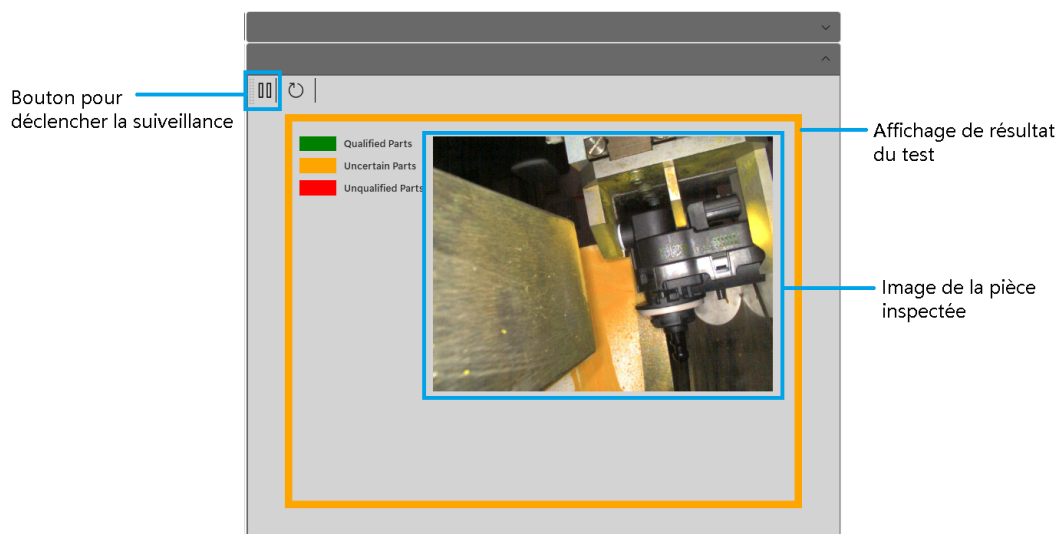


FIGURE 12 – Interface graphique de la surveillance de la ligne d'assemblage

Les pièces dont le score est inférieur à 90 sont considérées comme non qualifiées, celles dont le score est supérieur ou égal à 95 sont considérées comme qualifiées, et celles dont le score est compris entre 90 et 95 doivent être testées à nouveau manuellement. La réalisation de la fonctionnalité utilise la même approche que dans la section précédente, en utilisant un `thread` et `FileWatcher` pour assurer l'interaction de l'interface.

Sauvegarde des données de pièces

Une fois la pièce analysée, ses données sont sauvegardées dans une autre interface. Dans cette interface, j'ai conçu trois listes pour stocker les données des différents types de pièces, y compris son temps de production, le nombre de points gagnés et le chemin où l'image est sauvegardée.

Qualified Parts History		Uncertain Parts History		Unqualified Parts History	
Production Time	Score Path	Production Time	Score Path	Production Time	Score Path
2023/7/26 16:04:29	97 C:\Pc	2023/7/26 16:08:17	92 C:\Pow	2023/7/26 16:08:12	36 C:\Pow
2023/7/26 16:04:34	97 C:\Pc			2023/7/26 16:08:22	0 C:\Pow
2023/7/26 16:04:39	96 C:\Pc				
2023/7/26 16:04:44	96 C:\Pc				
2023/7/26 16:04:50	96 C:\Pc				
2023/7/26 16:04:55	97 C:\Pc				
2023/7/26 16:05:00	97 C:\Pc				
2023/7/26 16:05:20	97 C:\Pc				
2023/7/26 16:05:25	97 C:\Pc				
2023/7/26 16:05:30	97 C:\Pc				
2023/7/26 16:05:35	96 C:\Pc				
2023/7/26 16:05:40	97 C:\Pc				
2023/7/26 16:05:50	97 C:\Pc				
2023/7/26 16:05:55	97 C:\Pc				
2023/7/26 16:06:00	97 C:\Pc				
2023/7/26 16:06:05	97 C:\Pc				
2023/7/26 16:06:10	97 C:\Pc				
2023/7/26 16:08:01	97 C:\Pc				

Type de données stockées

Stockage séparé des pièces pour différentes situations

FIGURE 13 – Interface graphique pour le stockage des données relatives aux pièces

Et pour les pièces dont la qualification n'est pas certaine, vous pouvez les vérifier manuellement en cliquant sur la liste. Une nouvelle fenêtre apparaît alors et l'état de la pièce peut être vérifié en faisant glisser l'image vers la gauche ou la droite.

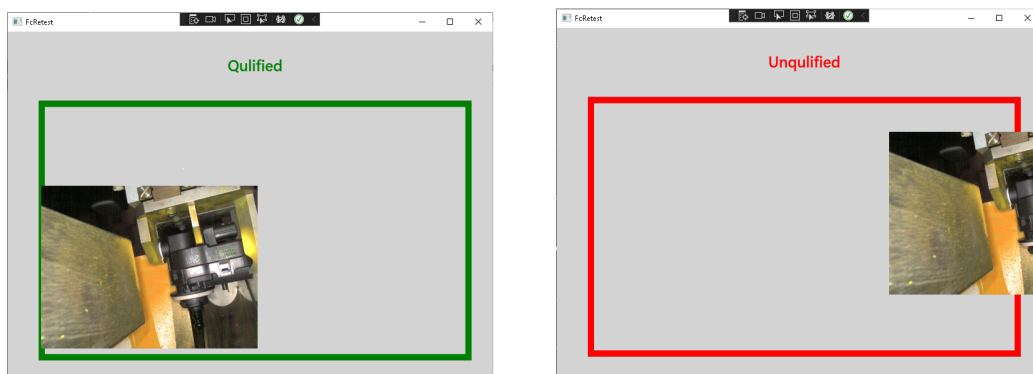


FIGURE 14 – Interface graphique pour la vérification manuelles

3.2.4 Suivi des données relatives aux pièces

Le module est conçu pour une utilisation à long terme et retracer les données de la pièce précédente enregistrées dans des fichiers Excel.

Interroger les données historiques

L'idée de cette fonction est d'afficher le fichier Excel sélectionné dans l'interface, d'effectuer une recherche par mot-clé et d'enregistrer les résultats de la recherche.

Dans la conception de l'interface, J'utilise une grille de données pour afficher les données d'un fichier Excel, une barre de sélection pour choisir le type de données à filtrer et une zone de texte pour saisir des mots-clés.

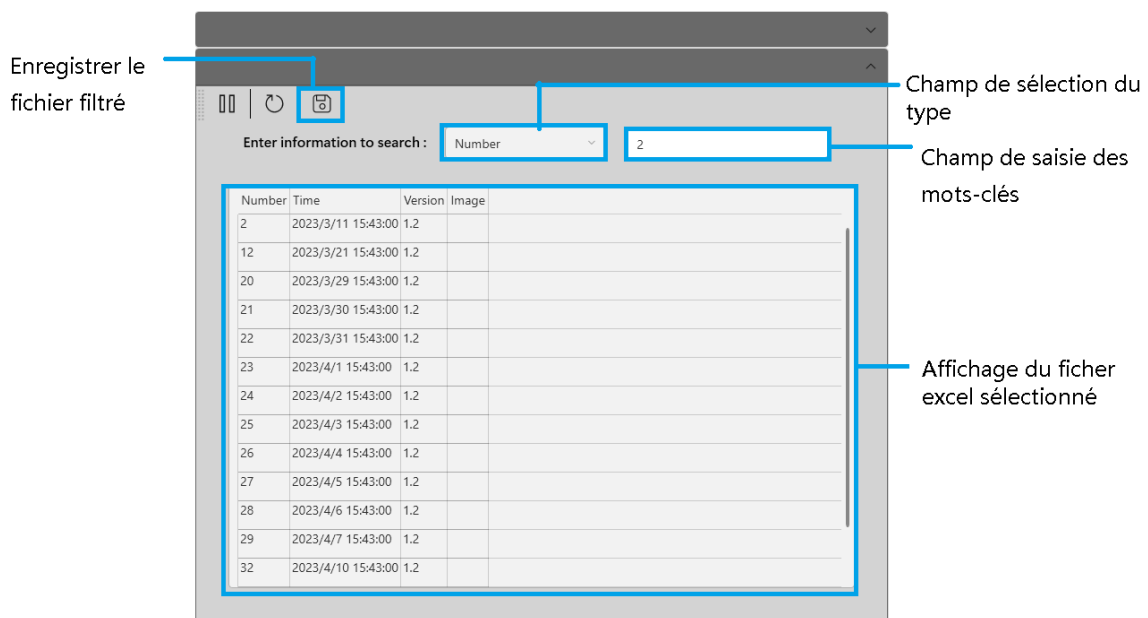


FIGURE 15 – Interface graphique pour l'interrogation de l'historique

Afin de lire le fichier Excel, une extension **ClosedXML** est utilisée, ce qui me permet d'obtenir le contenu du fichier par chemin et de le stocker dans la classe **DataTable**. J'ai ainsi créé deux fonctions pour convertir un chemin d'accès en **DataView** contenant un tableau et un chemin d'accès en une liste contenant les lignes d'en-tête d'un tableau. Et pour obtenir cette fonction de filtrage, la propriété **RowFilter** de **DataView** est tout à fait appropriée, il suffit de changer sa valeur en fonction de différents mots-clés pour modifier le contenu de l'affichage. Enfin, pour pouvoir enregistrer le tableau filtré, il est nécessaire de créer une classe **DataView** qui supprime effectivement les données en fonction du **RowFilter** et de la convertir en utilisant l'extension.

3.2.5 Configuration des données globales

Le module est principalement conçu pour permettre aux développeurs de compléter la sélection de la configuration de plusieurs modules directement par l'importation de fichiers de configuration afin de faciliter les tests. Par ailleurs, dans les scénarios d'application pratique, la construction de l'environnement d'exécution peut également être facilitée par une configuration globale.

Affichage des informations sur la configuration actuelle

L'interface est conçue pour permettre aux développeurs de visualiser la configuration globale actuelle du logiciel. Dans la section précédente de l'introduction générale, j'ai expliqué la structure de données qui compose **PowerEye**, consistant en les listes d'exemples de composants pour chaque interface de chaque module. Je n'ai donc besoin que d'afficher cette classe de données sur l'interface. Dans le choix de la méthode, j'ai choisi d'afficher le contenu après la sérialisation **XML**, car cela convient mieux à ma réalisation ultérieure de la fonction d'importation et d'exportation, d'autre part, c'est plus pratique que de reconstruire toute la fonction `toString` de données. En outre, j'ai développé la fonctionnalité de sérialisation afin de sauvegarder les configurations actuelles.

En termes de conception de l'interface graphique, cette section est plutôt simple, avec une zone de texte pour afficher les informations de configuration et un bouton pour sauvegarder la configuration actuelle.



FIGURE 16 – Interface graphique montrant la configuration globale

Importation de configuration

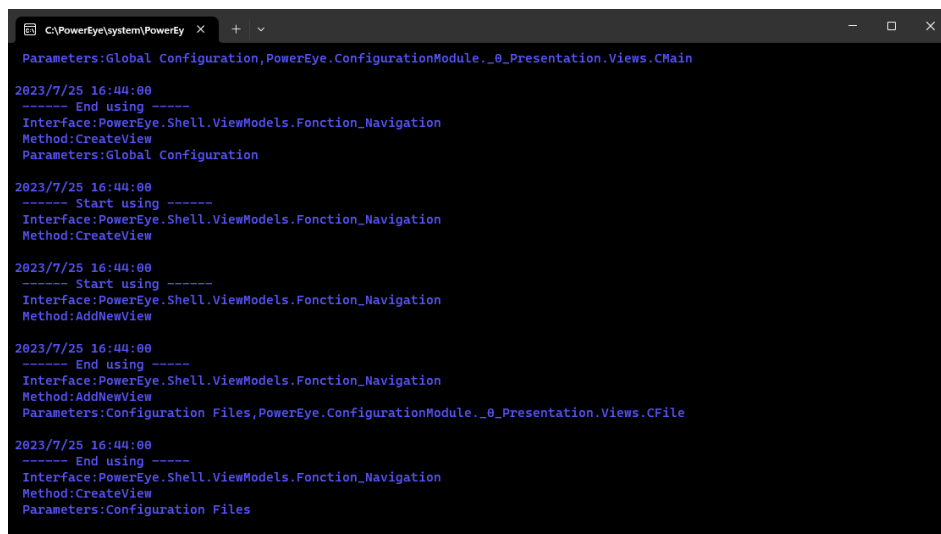
Cette interface est conçue pour appliquer les configurations précédemment sauvegardées à l'ordinateur. En utilisant la désérialisation **XML**, nous pouvons reconstruire le fichier sauvegardé en classe de données de **PowerEye**. Dans la classe de données, j'ai implémenté une fonction de copie pour chaque sous-classe afin de remplacer la valeur par la valeur générée par le fichier.

3.2.6 Flux de travail personnalisés

3.3 Fonctions en dehors du corps principal

Dans la dernière partie de mes réalisations, je vais présenter deux fonctionnalités que j'ai développées en dehors du corps principal du logiciel. Bien que ces deux fonctions ne constituent pas directement le logiciel, elles contribuent au processus de tests.

3.3.1 Affichage de l'appel de fonction



```
C:\PowerEye\system\PowerEye
Parameters:Global Configuration,PowerEye.ConfigurationModule._0_Presentation.Views.CMain

2023/7/25 16:44:00
----- End using -----
Interface:PowerEye.Shell.ViewModels.Fonction_Navigation
Method:CreateView
Parameters:Global Configuration

2023/7/25 16:44:00
----- Start using -----
Interface:PowerEye.Shell.ViewModels.Fonction_Navigation
Method:CreateView

2023/7/25 16:44:00
----- Start using -----
Interface:PowerEye.Shell.ViewModels.Fonction_Navigation
Method:AddNewView

2023/7/25 16:44:00
----- End using -----
Interface:PowerEye.Shell.ViewModels.Fonction_Navigation
Method:AddNewView
Parameters:Configuration Files,PowerEye.ConfigurationModule._0_Presentation.Views.CFile

2023/7/25 16:44:00
----- End using -----
Interface:PowerEye.Shell.ViewModels.Fonction_Navigation
Method:CreateView
Parameters:Configuration Files
```

FIGURE 17 – l'interface graphique d'ajout du bruit

3.3.2 Exécution de la chaîne de caractères comme un script

4. Conclusion