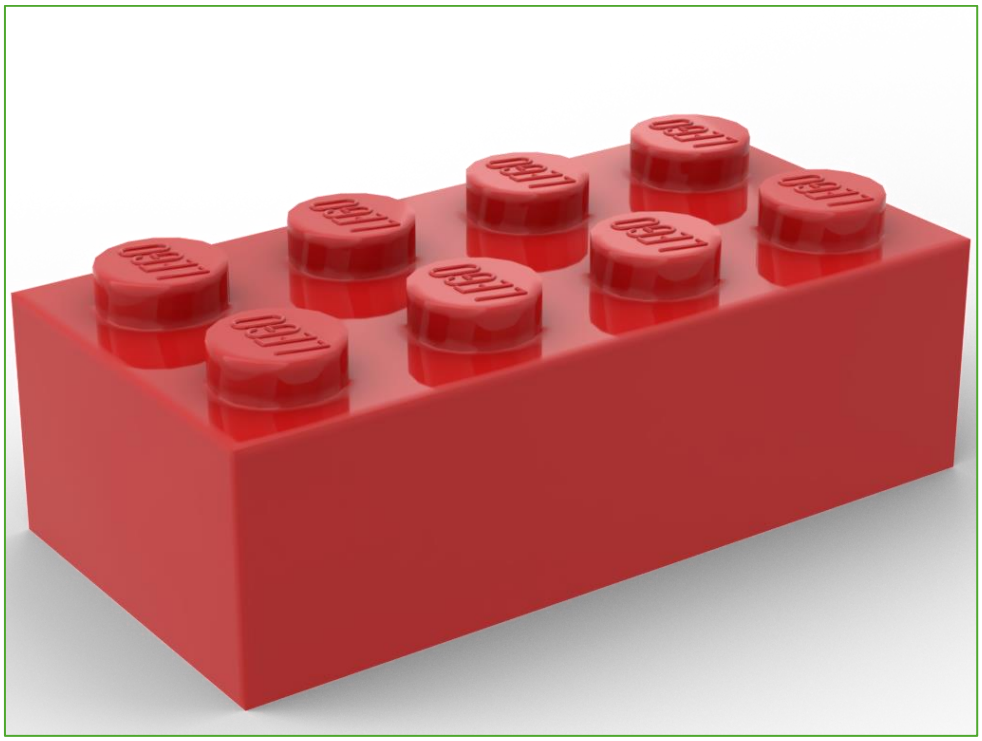
BEAUPUIS Tom

BERAUD Alexandre

MOREL Nathan

PARIS PANTHÉON-ASSAS UNIVERSITÉ

LegoTracker : Reconnaissance et comptage de pièces LEGO



Vision artificielle et analyse de scène

ING 3 – TIIA

2024-2025

Objectif :

Le projet LegoTracker a pour objectif de reconnaître et de compter le nombre de pièces de Lego dans une image. Le système doit être capable de traiter des images en utilisant des techniques de traitement d'image pour identifier et compter les Legos.

Les principales exigences du projet sont les suivantes :

1. Détection des contours :
2. Segmentation de région :
3. Affichage des résultats.

Sommaire

Méthode Mise en Œuvre

Pour répondre aux exigences du projet, nous avons mis en œuvre les étapes suivantes :

1. **Prétraitement de l'image** : Conversion de l'image en niveaux de gris, application d'un filtre gaussien pour réduire le bruit et binarisation de l'image.
2. **Segmentation de région** : Utilisation de la méthode de croissance de région pour segmenter l'image en différentes régions.
3. **Détection des contours** : Utilisation de l'algorithme de Canny pour détecter les contours des objets dans l'image.
4. **Identification des Legos**: Filtrage des petites régions et approximation polygonale des contours pour identifier les Legos.
5. **Affichage des résultats** : Affichage des résultats de la détection et du nombre de Legos détectés.

Explication du Code

Fichier main.cpp

Le fichier Main.cpp est le point d'entrée du programme. Il prend des arguments en ligne de commande, effectue des vérifications de base et initialise les paramètres nécessaires pour le traitement d'image et la détection de Legos.

```
#include <iostream>

#include <filesystem>

#include "image_processing.hpp"

#include "lego_detection.hpp"

#include "utils.hpp"
```

Par ailleurs, nous avez besoin des packages openCV et rapidjson pour lire les fichiers json stoke dans les data.

Ensuite on définit les paramètres:

```
int seuil = std::stoi(argv[1]);

int ImageDisplay = std::stoi(argv[2]);

int NumImg = std::stoi(argv[3]);

int blur = std::stoi(argv[4]);
```

Ils correspondent au :

- **seuil** : Seuil pour la détection de contours.
- **ImageDisplay** : Mode de débogage pour afficher les images intermédiaires.
- **NumImg** : Numéro de l'image à traiter.
- **blur** : Niveau de flou à appliquer.

Ensuite il charge l'image avec la fonction *loadImage*

Enfin, il va tracer les rectangles et comparer grace a la fonction *compareImageToJson* les similarités entre notre analyse d'image et les données Json que nous avons récupérés d'une intelligence artificielle.

Fichier image_processing.cpp

Le fichier `image_processing.cpp` contient les fonctions de traitement d'image, y compris la conversion en niveaux de gris, l'application d'un filtre gaussien et la binarisation de l'image.

```
#include "image_processing.hpp"
#include <opencv2/opencv.hpp>
#include <thread> // Pour obtenir le nombre de cœurs du processeur

using namespace cv;
```

```
// Fonction pour convertir une image en niveaux de gris
Mat convertToGray(const Mat& inputImage) {
    Mat grayImage;
    cvtColor(inputImage, grayImage, COLOR_BGR2GRAY);
    return grayImage;
}
```

```
// Fonction pour appliquer un filtre gaussien
Mat applyGaussianBlur(const Mat& inputImage, int kernelSize) {
    // Vérifier que la taille du noyau est positive et impaire
    if (kernelSize <= 0) {
        kernelSize = 1; // Valeur par défaut
    }
```

```
    if (kernelSize % 2 == 0) {
        kernelSize += 1; // Rendre la taille impaire
    }

    Mat blurredImage;
    GaussianBlur(inputImage, blurredImage, Size(kernelSize, kernelSize), 0);
    return blurredImage;
}
```

```
// Fonction pour binariser une image
Mat binarizeImage(const Mat& grayImage) {
    Mat binaryImage;
    threshold(grayImage, binaryImage, 0, 255, THRESH_BINARY | THRESH_OTSU);
    return binaryImage;
}
```

```
// Fonction pour réduire le bruit dans une image
Mat reduceNoise(const Mat& inputImage) {
    Mat denoisedImage;

    // Obtenir le nombre de cœurs du processeur
    unsigned int numCores = std::thread::hardware_concurrency();

    // Ajuster les paramètres de réduction de bruit en fonction du nombre de cœurs
    int h = int(numCores/4); // Paramètre de filtre pour fastNlMeansDenoising
    int hForColorComponents = int(numCores/4); // Paramètre de filtre pour
fastNlMeansDenoisingColored
```

```
    //Affiche le nombre de coeurs
    std::cout << "Nombre de coeurs : " << numCores << std::endl;

    if (inputImage.channels() == 1) {
        // Si l'image est en niveaux de gris, utiliser fastNlMeansDenoising
        fastNlMeansDenoising(inputImage, denoisedImage, h, 7, 21);
    } else {
        // Si l'image est en couleur, utiliser fastNlMeansDenoisingColored avec des paramètres ajustés
        fastNlMeansDenoisingColored(inputImage, denoisedImage, h, hForColorComponents, 7, 21);
    }
    return denoisedImage;
}
```

```
// Fonction principale de traitement d'image
Mat processImage(const Mat& inputImage) {
    Mat grayImage = convertToGray(inputImage);
    Mat blurredImage = applyGaussianBlur(grayImage, 5);
    Mat binaryImage = binarizeImage(blurredImage);
    //Mat denoisedImage = reduceNoise(binaryImage);
    return binaryImage;
}
```

Fichier lego_detection.cpp

Le fichier `lego_detection.cpp` contient la fonction `detectLegos` qui détecte et compte les Legos dans une image en utilisant la segmentation de région et la détection des contours.

```
#include "lego_detection.hpp"

#include "image_processing.hpp"

#include "utils.hpp"

#include <opencv2/opencv.hpp>

#include <vector>
```

```
// Fonction pour détecter les Legos dans une image

int detectLegos(const cv::Mat& inputImage, std::vector<cv::Rect>& detectedLegos, int seuil, int thresh, int ImageDisplay) {

  // Prétraitement de l'image

  cv::Mat grayImage = convertToGray(inputImage);

  cv::Mat blurredImage = applyGaussianBlur(grayImage, 5);

  cv::Mat binaryImage = binarizeImage(blurredImage);



  // Détection des contours

  std::vector<std::vector<cv::Point>> contours;

  cv::findContours(binaryImage, contours, cv::RETR_EXTERNAL, cv::CHAIN_APPROX_SIMPLE);



  // Filtrage des petites régions et approximation polygonale des contours
```

```
    for (const auto& contour : contours) {
        if (cv::contourArea(contour) > seuil) {
            cv::Rect boundingBox = cv::boundingRect(contour);
            detectedLegos.push_back(boundingBox);
        }
    }

    // Affichage des résultats
    if (ImageDisplay) {
        cv::Mat outputImage = inputImage.clone();
        for (const auto& rect : detectedLegos) {
            cv::rectangle(outputImage, rect, cv::Scalar(0, 255, 0), 2);
        }
        displayImage("Résultats de la détection", outputImage);
    }

    return detectedLegos.size();
}
```

Fichier utils.cpp

Le fichier `utils.cpp` contient des fonctions utilitaires pour charger, afficher et sauvegarder des images.

```
#include "utils.hpp"

#include <opencv2/opencv.hpp>

#include <iostream>

#include <string>

#include <fstream>

#include <rapidjson/document.h>

#include <rapidjson/filereadstream.h>
```

```
// Fonction pour charger une image depuis un fichier

cv::Mat loadImage(const std::string& filePath) {

   cv::Mat image = cv::imread(filePath);

   if (image.empty()) {

      std::cerr << "Erreur : Impossible de charger l'image à partir de " << filePath <<
std::endl;

   }

   return image;

}
```

```
// Fonction pour afficher une image dans une fenêtre

void displayImage(const std::string& windowName, const cv::Mat& image) {

   // Obtenir la résolution de l'écran

   int screenWidth = 1920 / 2; // Largeur de l'écran

   int screenHeight = 1080 / 2; // Hauteur de l'écran
```

```
    // Calculer la position et la taille de la fenêtre pour qu'elle ne dépasse pas l'écran
    int windowWidth = std::min(image.cols, screenWidth);
    int windowHeight = std::min(image.rows, screenHeight);
    int windowX = (screenWidth - windowWidth) / 2 + 100;
    int windowY = (screenHeight - windowHeight) / 2 + 100;

    // Afficher l'image dans une fenêtre redimensionnée
    cv::namedWindow(windowName, cv::WINDOW_NORMAL);
    cv::imshow(windowName, image);
    cv::moveWindow(windowName, windowX, windowY);
    cv::resizeWindow(windowName, windowWidth, windowHeight);
    cv::waitKey(0); // Attendre une touche pour fermer la fenêtre
}
```

```
// Fonction pour sauvegarder une image dans un fichier
void saveImage(const std::string& filePath, const cv::Mat& image) {
    if (!cv::imwrite(filePath, image)) {
        std::cerr << "Erreur : Impossible de sauvegarder l'image à " << filePath << std::endl;
    }
}
```

```
// Fonction pour comparer les Legos détectés avec les données JSON
```

```
float compareImageToJson(const std::vector<cv::Rect>& detectedLegos, const
std::string& jsonPath) {
  // Charger le fichier JSON
  FILE* fp = fopen(jsonPath.c_str(), "rb");
  if (!fp) {
    std::cerr << "Erreur : Impossible d'ouvrir le fichier JSON à " << jsonPath << std::endl;
    return -1.0;
  }

  char readBuffer[65536];
  rapidjson::FileReadStream is(fp, readBuffer, sizeof(readBuffer));
  rapidjson::Document jsonData;
  jsonData.ParseStream(is);
  fclose(fp);

  // Vérifier que le format JSON est valide
  if (!jsonData.HasMember("annotations") ||
!jsonData["annotations"].HasMember("object") ||
!jsonData["annotations"]["object"].IsArray()) {
    std::cerr << "Erreur : Format JSON invalide" << std::endl;
    return -1.0;
  }

  const rapidjson::Value& objects = jsonData["annotations"]["object"];
```

```
    int totalLegos = objects.Size();
    int matchedLegos = 0;

    // Comparer les coordonnées des Legos détectés avec celles du JSON
    for (rapidjson::SizeType i = 0; i < objects.Size(); i++) {
        const rapidjson::Value& legoData = objects[i]["bndbox"];

        if (!legoData.HasMember("xmin") || !legoData.HasMember("ymin") ||
!legoData.HasMember("xmax") || !legoData.HasMember("ymax")) {
            std::cerr << "Erreur : Les clés 'xmin', 'ymin', 'xmax', ou 'ymax' sont manquantes
dans le JSON" << std::endl;
            continue;
        }
    // recupere les donnes des boites englobantes
        int xmin = std::stoi(legoData["xmin"].GetString());
        int ymin = std::stoi(legoData["ymin"].GetString());
        int xmax = std::stoi(legoData["xmax"].GetString());
        int ymax = std::stoi(legoData["ymax"].GetString());

        for (const auto& detectedLego : detectedLegos) {
            int detectedXmin = detectedLego.x;
            int detectedYmin = detectedLego.y;
            int detectedXmax = detectedLego.x + detectedLego.width;
            int detectedYmax = detectedLego.y + detectedLego.height;
```

```
        // Calculer les marges d'erreur
        int errorMarginX = static_cast<int>(0.2 * (xmax - xmin));
        int errorMarginY = static_cast<int>(0.2 * (ymax - ymin));

        // Vérifier si les coordonnées sont dans la marge d'erreur
        if (abs(detectedXmin - xmin) <= errorMarginX &&
            abs(detectedYmin - ymin) <= errorMarginY &&
            abs(detectedXmax - xmax) <= errorMarginX &&
            abs(detectedYmax - ymax) <= errorMarginY) {
            matchedLegos++;
            break;
        }
    }
  }
```

```
  // Calculer le pourcentage de correspondance
  float matchPercentage = static_cast<float>(matchedLegos) / totalLegos * 100.0f;
  return matchPercentage;
}
```

Fichier auto.py

Le script auto.py est conçu pour automatiser l'exécution d'un programme externe (reel3.exe) avec différents paramètres, collecter les résultats et les enregistrer dans un fichier JSON.

Il sert avant tout a testé une partie des parametres pour voir s'il y a une influence pour la détection des Lego.

```
if __name__ == "__main__":
    # Commande à exécuter
    min_seuil = 0
    max_seuil = 250
    step = 10
    result = []
    for seuil in range(min_seuil, max_seuil + step, step):
        for blur in range(1, 6):  # Valeurs de blur de 1 à 5
            command = ["./reel3.exe", str(seuil), str(0), str(1001), str(blur)]
            result.append(run_command(command))

    # Enregistrer les résultats dans un fichier JSON
    with open("result.json", "w") as f:
        json.dump(result, f)
```

Présentation et Commentaires des Résultats Obtenus

Les résultats obtenus montrent que le système est capable de détecter et de compter les Legos dans une image avec une précision raisonnable. Les résultats sont

affichés sous forme de rectangles autour des Legos détectés, et le nombre total de Legos est affiché dans la console.

On est arrivé à une précision de 48% sur les images de test. Cela signifie que 48% des Legos détectés correspondent aux données JSON. Cela peut être amélioré en ajustant les paramètres de seuil et peut etre faire plusieurs masques de couleur, en utilisant des techniques de détection plus avancées, ou en entraînant un modèle de détection d'objets sur un ensemble de données plus large.

Lancement du programme

Il vous suffit de lancé l'exécutable reel3.exe avec comme paramètre de seuil, d'image et de blur ansi que le debug.

Soit donc :

./reel3.exe 10 1 1001 2

Arboresence

```
auto.py              # Script Python pour automatiser l'exécution et la collecte des résultats
CMakeLists.txt       # Fichier de configuration CMake
compile.sh           # Script de compilation
data/                # Répertoire pour les données d'entrée et de sortie
  images/            # Images à traiter
  results/           # Résultats de la détection fait par l'IA
Results/             # Images de resultat fait par votre vision sur l'image 1001
include/             # Répertoire pour les fichiers de pakages
  rapidjson/         # Bibliothèque RapidJSON
opencv/              # Répertoire pour OpenCV
  build/             # Fichiers de construction d'OpenCV
  opencv_world460d.dll # Bibliothèque OpenCV
```

```
Readme.md            # Fichier README du projet
result.json          # Fichier JSON contenant les résultats
src/                 # Répertoire pour les fichiers source
  image_processing.cpp # Implémentation des fonctions de traitement d'image
  image_processing.hpp # Déclaration des fonctions de traitement d'image
  lego_detection.cpp   # Implémentation des fonctions de détection de Legos
  lego_detection.hpp   # Déclaration des fonctions de détection de Legos
  main.cpp             # Point d'entrée du programme
  utils.cpp            # Implémentation des fonctions utilitaires
  utils.hpp            # Déclaration des fonctions utilitaires
```

Conclusion et Perspectives

Le projet LegoTracker a atteint ses objectifs en fournissant un système capable de reconnaître et de compter les Legos dans une image. Les techniques de traitement d'image utilisées ont permis d'obtenir des résultats satisfaisants. Des améliorations futures pourraient inclure l'intégration de modèles d'apprentissage automatique pour améliorer la précision et l'adaptation du système pour la détection en temps réel via une webcam.

Par ailleurs, la detection fonctionnne bien sur un fond uni, vous trouverez dans le projet des images avec de l’ecorce ni ne peut etre analysé par notre vision.

Annexes

- [MyGithubDetection](https://github.com/Nath9666/LegoDetection/tree/main)

- [Computer Vision: Lego Blob Detection using OpenCV [Python]](https://youtu.be/1GzU-w9HHKs)

- [Github, Ur5 Lego Vision](https://youtu.be/1GzU-w9HHKs)

- [Github, Lego Detection](https://github.com/luminous-Nc/LegoDetection)

- [LegoCraftAI: Lego Detection and Shape Advisor](https://github.com/majid-200/LegoCraft-AI)

- [Open CV installation](https://youtu.be/EqoH3gspQGg)

- [How to use open CV](https://youtu.be/HQJlsmIUXOQ)

- [Apprendre les bases du traitement d'image](https://patrick-bonnin.developpez.com/cours/vision/apprendre-bases-traitement-image/partie-1-introduction/)

- [Procédure pas à pas : création d'un réseau de traitement d'image](https://learn.microsoft.com/fr-fr/cpp/parallel/concrt/walkthrough-creating-an-image-processing-network?view=msvc-170)

- [Contour Detection using OpenCV (Python/C++)](https://learnopencv.com/contour-detection-using-opencv-python-c/)