Rapport de Projet: LegoTracker

Ce projet a été réalisé par **Nathan MOREL**, **Alexandre BÉRAUD** et **Tom BEAUPUIS**

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. **Reconnaissance des couleurs** : Utiliser la décomposition HSV pour détecter les couleurs principales des Legos.
2. **Détection des contours** : Utiliser l'algorithme de Canny pour détecter les contours des Legos.
3. **Segmentation de région** : Utiliser la méthode de croissance de région pour segmenter l'image en différentes régions.
4. **Affichage des résultats** : Afficher les résultats de la détection et du nombre de Legos détectés.

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` contient le point d'entrée principal du programme. Il lit les arguments de seuil et de seuil de l'utilisateur, charge l'image, détecte les Legos et affiche les résultats.

```
#include <iostream>
#include "image_processing.hpp"
#include "lego_detection.hpp"
#include "utils.hpp"

int main(int argc, char** argv) {
    if (argc < 4) {
        std::cerr << "Usage: " << argv[0] << " <seuil> <thresh>" << std::endl;
        return -1;
    }

    int seuil = std::stoi(argv[1]);
    int thresh = std::stoi(argv[2]);
    int ImageDisplay = std::stoi(argv[3]);
    if (seuil <= 0 || thresh <= 0) {
        std::cerr << "Erreur : Les valeurs de seuil et de thresh doivent être
positives." << std::endl;
        return -1;
    }

    if (ImageDisplay != 0 && ImageDisplay != 1) {
        std::cerr << "Erreur : La valeur de ImageDisplay doit être 0 ou 1." <<
std::endl;
        return -1;
    }
```

```
    cv::Mat image = loadImage("./data/images/1001.png");

    // Détecter les Legos
    std::vector<cv::Rect> detectedLegos;
    int legoCount = detectLegos(image, detectedLegos, seuil, thresh, ImageDisplay);

    // Afficher les résultats de la détection
    displayDetectionResults(image, detectedLegos);

    // Comparer les résultats avec les données JSON
    float similarity = compareImageToJson(detectedLegos,
"./data/results/1001.json");
    if (similarity >= 0.0) {
        std::cout << "Similarité : " << similarity << std::endl;
    }

    // Afficher le nombre de Legos détectés
    std::cout << "Nombre de Legos : " << legoCount << std::endl;

    // Attendre une touche pour quitter
    cv::waitKey(0);

    return EXIT_SUCCESS;
}
```

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);
    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.

```cpp
#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;
        }

        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>(0.1 * (xmax - xmin));
            int errorMarginY = static_cast<int>(0.1 * (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 fichier `auto.py` exécute le programme avec différentes valeurs de seuil et de seuil, et enregistre les résultats dans un fichier JSON. Il sert également à automatiser la comparaison entre les résultats obtenus et les résultats attendus.

```
import subprocess
import json

def run_command(command):
    try:
        # Exécuter la commande
        result = subprocess.run(command, capture_output=True, text=True,
check=True)
        # Afficher la sortie standard
        for line in result.stdout.splitlines():
            if "Nombre de Legos" in line:
                print("Nombre de Legos :", line)
                nb_lego = line.split(":")[1].strip()
        print(nb_lego, command[1], command[2])
        # Afficher les erreurs s'il y en a
        if result.stderr:
            print("Errors:\n", result.stderr)
    except subprocess.CalledProcessError as e:
        print(f"An error occurred: {e}")
    return {"nb_lego": nb_lego, "seuil": command[1], "thresh": command[2]}

if __name__ == "__main__":
    # Commande à exécuter
    min_seuil = 0
    max_seuil = 250
    step = 10
    min_thresh = 0
    max_thresh = 250
    result = []
    for seuil in range(min_seuil, max_seuil+step, step):
        for thresh in range(min_thresh, max_thresh+step, step):
            command = ["./visionArti.exe", str(seuil), str(thresh)]
            result.append(run_command(command))

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

Fichier xmltoDict.py

Le fichier `xmltoDict.py` convertit les fichiers XML dans le dossier results en fichiers JSON et les enregistre dans le même dossier. Nous avions pris une base de données mais malheureusement les données étaient en XML et nous avons donc dû les convertir en JSON pour pouvoir les comparer avec nos résultats.

```
import xmltodict
import json
import os

# Chemin du dossier contenant les fichiers XML
input_folder = 'c:/Users/Nathan/Documents/Projet/visionArti/data/results'

# Parcourir tous les fichiers dans le dossier
for filename in os.listdir(input_folder):
    if filename.endswith('.xml'):
        # Chemin complet du fichier XML
        xml_file_path = os.path.join(input_folder, filename)

        # Lire le fichier XML
        with open(xml_file_path, 'r') as file:
            xml_content = file.read()

        # Convertir le XML en dictionnaire
        xml_dict = xmltodict.parse(xml_content)

        # Convertir le dictionnaire en JSON
        json_content = json.dumps(xml_dict, indent=4)

        # Chemin complet du fichier JSON de sortie
        json_file_path = os.path.join(input_folder, filename.replace('.xml',
'.json'))

        # Écrire le contenu JSON dans le fichier de sortie
        with open(json_file_path, 'w') as json_file:
            json_file.write(json_content)
```

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 35% sur les images de test. Cela signifie que 35% des Legos détectés correspondent aux données JSON. Cela peut être amélioré en ajustant les paramètres de seuil et de thresh, en utilisant des techniques de détection plus avancées, ou en entraînant un modèle de détection d'objets sur un ensemble de données plus large.

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.

Annexes

- MyGithubDetection
- Computer Vision: Lego Blob Detection using OpenCV [Python]
- Github, Ur5 Lego Vision
- Github, Lego Detection
- LegoCraftAI: Lego Detection and Shape Advisor
- Open CV installation
- How to use open CV
- Apprendre les bases du traitement d'image
- Procédure pas à pas : création d'un réseau de traitement d'image
- Contour Detection using OpenCV (Python/C++)

DataSets

- DataSet LegoBricks OnebyOne
- DataSet LegoBricks OnebyOne 3D
- DataSet LegoBricks OnebyOne reel

- DataSet LegoBricks Vrack reel