Final Project Report

Group 4 - 張紀睿 312551086 / 翁堉豪 312551076

Introduction

本次 final project 中，實作了在嵌入式系統上進行物件偵測的程式。
為了確保在嵌入式系統上的執行效率，我們採用了 Canny 邊緣偵測演算法來尋找物件，並根據物件的形狀、大小判斷偵測到的物件類別，以此方式實現了接近 real-time 的物件偵測。

環境配置

- OS: Ubuntu 14.04
- Compiler: arm-linux-gnueabihf-g++
- Library: opencv3.4.7
- Programming Language: C++

實作方法

Step 1 邊緣檢測

將影像轉化為灰階影像，並利用 cv::Canny 進行取得影像的邊緣。

```
cv::cvtColor(img, gray, cv::COLOR_BGR2GRAY);
cv::Canny(gray, edge, 100, 200); //hyperparameter
```

Step 2 尋找輪廓

透過 cv::findContours 取得邊緣中的輪廓以利後續使用

```
cv::findContours(edge, contours, cv::RETR_TREE,
  cv::CHAIN_APPROX_SIMPLE);
```

Step 3 從輪廓中獲取物件資訊

將每個輪廓視為一個物件依序處理，根據 boundingBox 計算出物件大小以及物體的長寬比例。

```
for (size_t i = 0; i < contours.size(); i++) {
        cv::Rect boundingBox = cv::boundingRect(contours[i]);
        float width = boundingBox.width;
        float height = boundingBox.height;
        float ratio;
        if (width > height) {
            ratio = width / height;
        }
        else {
            ratio = height / width;
        }
        int rectArea = boundingBox.width * boundingBox.height;
                // ...remaining code
     }
```

Step 4 偵測類別

讀取包含各個 class 大小與比例的設定檔 config.txt 並與物件資訊比較，若有相符的 class 則畫出 label 與 bounding box。

```
for (size_t i = 0; i < contours.size(); i++) {
        // previous code
        std::ifstream configFile("config.txt");
        int min_area, max_area;
        float min_ratio, max_ratio;
        std::string label;
        // 使用 while 迴圈逐行讀取檔案
        while (configFile >> label >> min_area >>
            max_area >> min_ratio >> max_ratio) {
            if (rectArea >= min_area && max_area >= rectArea
              && ratio >= min_ratio && max_ratio >= ratio) {
```

```
                cv::rectangle(img, boundingBox,
                  cv::Scalar(0, 255, 0), 2);

                cv::putText(img, label,
                  cv::Point(boundingBox.x, boundingBox.y - 5),
                  cv::FONT_HERSHEY_SIMPLEX, 0.5,
                  cv::Scalar(0, 255, 0), 2);
            }
        }
        configFile.close();
    }
```

Step 5 儲存結果/顯示結果

若是進行圖片的偵測，則將預測結果儲存於 result.png後結束執行。

```
cv::imwrite("result.png", img);
cv::Mat image;
```

若進行即時偵測，則將畫面顯示於framebuffer，並等待下一個frame。

檔案執行

編譯 use_config_detect.cpp 與 real_time.cpp 後會產生其對應的兩個執行檔。另外，執行時會需要設置一個 config.txt。

檔案列表

- use_config_detect: 針對輸入圖片進行物件偵測。
- real_time: 利用 camera 拍攝畫面進行即時偵測。
- config.txt: 每行為一個class，以 Label minArea maxArea minRatio maxRatio 為順序。

執行命令

將執行檔與 config.txt 及 opencv 之 library 放在同一個資料夾，並執行以下命令

```
//圖片偵測
LD_LIBRARY_PATH=. ./use_config_detext
//即時偵測
LD_LIBRARY_PATH=. ./real_time {設定檔(optional，預設為config.txt)}
```

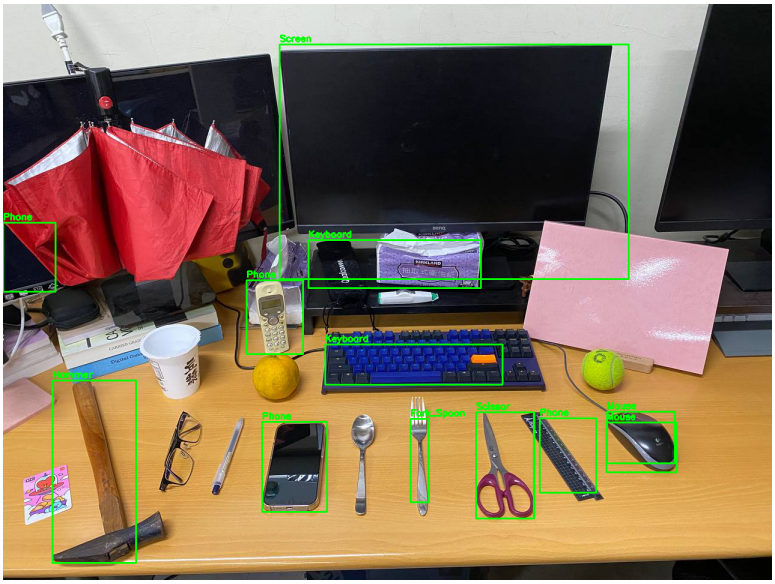
note. 請將本地端的的opencv library (libopencv_world.so.3.4.7) 改名為 libopencv_world.so.3.4 並置於開發板中欲使用程式之目錄。

Config 設定

每行為一個 class ，設定該類別的最小與最大面積，以及長寬比，符合大小與比例範圍的物件會被判斷為該類別，沒有數量設定的上限。

- Label: 類別的名稱，資料類型為 string。
- minArea: 類別的最小面積，資料類型為 int。
- maxArea: 類別的最大面積，資料類型為 int。
- minRatio: 類別的最小長寬比，資料類型為 float。
- maxRatio: 類別的最大長寬比，資料類型為 float。

實驗結果



遇到的問題

1. 運行時間: 由於在本次 project 中，運行時間是也是一項重要的指標，使用CNN等方式在嵌入式系統中常常最造成運算資源消耗的增加，而無法達到real-time，甚至需要很長的運算時間。

   解決方法

   我們決定採用傳統的邊緣檢測方法取代深度學習模型來進行物件偵測，大幅減少了運算所需的時間。

2. 物件偵測的準確性: 我們最初使用物體的大小作為判斷類別的依據，而僅使用大小會造成結果不準確以及在物體的角度、距離變化時無法有效預測。

   解決方法
   除了物體大小外，我們增加了藉由物體形狀(長寬比)作為判斷類別的另一項依據，並且可以以此來篩選掉形狀不合理的物件，以進一步增加準確性。

3. 類別數量: 我們希望建立的物件偵測程式可以根據需要調整需要偵測出來的物體種類與列表，而非固定的類別數量。

   解決方法

   將類別設定放置於config.txt中，再透過程式進行讀取，便可以更有彈性的進行設定。

Reference

- OpenCV Canny https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html