Fenerbahçe Üniversitesi BLM102 – Algoritmalar ve Programlama II

Görüntü İşleme Yöntemleri ile Üretim Hattı Analizi

Proje Ekibi

- Ahmet Hazar Haspolat
- Mustafa Berk Taşkın
- Alp Yılmaz
- Ömer Sait Yorulmaz

Fenerbahçe Üniversitesi

Projenin Amacı

Fabrikalarda üretim hatlarından çıkan ürünlerin analizi; yüksek çözünürlüklü ve yüksek FPS (Frame per Second) özellikli kameralar ile yapılmaktadır. Bir üretim hattından çıkan üç ürünün, video kaydı boyunca kaçar adet üretildiğinin tespit edileceği bir sistem geliştirilecektir.

Fenerbahçe Üniversitesi

```cpp
#include <opencv2/opencv.hpp>
#include <iostream>
#include<string>
#include"objectTracker.h"
#include"frameTracker.h"
int main() {

    cv::Mat orjFrame, copyFrame,result;
    cv::VideoCapture cap("C:\\Users\\OmerSait\\Desktop\\uretimHatti.mp4");
    if (!cap.isOpened()) {
        std::cout << "Error: Video cannot be opened" << std::endl;
        return -1;
    }
    cap.read(copyFrame);
    while (cap.isOpened()) {
        cap >> orjFrame;
        if (orjFrame.empty())
            break;

        cv::imshow("orjFrame", orjFrame);

        frameTracker frTracker(&orjFrame, &copyFrame);

        frTracker.toGray(CV_BGR2GRAY);

        frTracker.toBlur(cv::Size(9, 9));

        frTracker.toDif();

        frTracker.toTreshHolding(72, 255, CV_THRESH_BINARY);
        frTracker.toDilate();
        result = frTracker.toRemoveBackground();

        objectTracker objTracker;
        objTracker.setParams(0, 500, true, 300, false, 0.1, false, 0.87, false, 0.01);
        objTracker.toDetect(&result);
```

```
1  #pragma once
2  #include <opencv2/opencv.hpp>
3
4  class frameTracker
5  {
6  private:
7      cv::Mat orjFrame;
8      cv::Mat copyFrame;
9      cv::Mat difFrame;
10     cv::Mat result;
11     cv::Mat thresh;
12 public:
13     frameTracker(cv::Mat* orj, cv::Mat* copy);
14     void toGray(int code);
15     void toBlur(cv::Size ksize);
16     void toDif();
17     void toTreshHolding(double thresh, double maxval, int type);
18     void toDilate();
19     cv::Mat toRemoveBackground();
20 };
21
22
```

```cpp
#include "frameTracker.h"

frameTracker::frameTracker(cv::Mat* orj, cv::Mat* copy)
{
    this->orjFrame = *orj;
    this->copyFrame = *copy;

}

void frameTracker::toGray(int code)
{
    //A grayscale image only requires information of density.
    //All we need is one byte per pixel of grayscale image.
    // A byte can store a value between 0 and 255 so it contains all possible shades of gray
    // In some cases, its highly recommended that work with grayscale image.
    // Thats why that i convert this frames into grayscale frames.

    cv::cvtColor(this->orjFrame, this->orjFrame, CV_BGR2GRAY);
    cv::cvtColor(this->copyFrame, this->copyFrame, CV_BGR2GRAY);
}
```

```
21
22
23  void frameTracker::toBlur(cv::Size ksize)
24  {
25      //types of to blur
26      //blur - GaussianBlur - medianBlur - bilateralFilter
27      // There are several reason to use gaussian filter for this case. The Gaussian filter is a linear filter that can use -
28      // - them for "unsharp masking" (edge detection).
29      // Unlike the gaussian filter, median filter is nonlineer filter that removes noise while keeping edges relatively sharp.
30      // The video that i used for this project, the edges of product line must be blur as much as possible.
31
32      cv::GaussianBlur(this->orjFrame, this->orjFrame, ksize, 0);
33      cv::GaussianBlur(this->copyFrame, this->copyFrame, ksize, 0);
34  }
35
36
37  void frameTracker::toDif()
38  {
39      //Calculates the per-element absolute difference between two arrays or between an array and a scalar.
40      //I.e. calculates the differences between thw two frames.
41      //As a result of this extraction process, the changing parts, such as moving parts, are shown.
42      //The objective of the project is get some information on total count of each product.
43
44      cv::absdiff(this->orjFrame, this->copyFrame, this->difFrame);
45  }
46
47  void frameTracker::toTreshHolding(double thresh, double maxval, int type)
48  {
49      //Thresholding is typically used to get a bi-level (binary) image out of a grayscale image or or removing a noise
50      //Filtering out pixels can have too small or too large values
51      //Need to update the pixels on black or white according to the given threshold value.
52
53      threshold(this->difFrame, this->thresh, thresh, maxval, type);
54  }
55
56  void frameTracker::toDilate()
```

```
57
58  void frameTracker::toDilate()
59  {
60      //Increases the object area
61      //Used to accentuate features
62      // To dilate objects ensure the detection of any object as much as possible
63      // There are three types objects, two of which is in need of dilation
64
65      cv::Mat kernel = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(9, 9), cv::Point(-1, -1));
66      dilate(this->thresh, this->thresh, kernel);
67  }
68
69  cv::Mat frameTracker::toRemoveBackground()
70  {
71      //There are three ways to remove background
72      //BackgroundSubtractorMOG,  ve BackgroundSubtractorGMG
73      //BackgroundSubtractorMOG2 selects the appropriate number of Gaussian distributions for each pixel. That's why
74      // For more information = https://www.geeksforgeeks.org/background-subtraction-opencv/
75
76      cv::BackgroundSubtractorMOG2 mog2;
77      mog2(this->thresh, this->result);
78
79      return this->result;
80  }
81
82
```

```
1   #pragma once
2   #include <opencv2/opencv.hpp>
3   #include <vector>
4   #include "object.h"
5
6   class objectTracker
7   {
8   private:
9       cv::Mat frame;
10      std::vector<cv::KeyPoint> keypoints;
11      cv::SimpleBlobDetector::Params params;
12      std::vector<object> objectInfos;
13  public:
14      objectTracker();
15      void setParams(float minThreshold, float maxThreshold, bool filterByArea, float minArea,
16          bool filterByCircularity, float minCircularity, bool filterByConvexity,
17          float minConvexity, bool filterByInertia, float minInertiaRatio);
18      void toDetect(cv::Mat* result);
19  };
20
21
```

```cpp
#include "objectTracker.h"


objectTracker::objectTracker()
{
    object obj;

    // small objects
    obj.xAxis = 34;
    obj.yAxis = 218;
    obj.size = 16;
    objectInfos.push_back(obj);

    // medium objects
    obj.xAxis = 52;
    obj.yAxis = 201;
    obj.size = 30;
    objectInfos.push_back(obj);

    // large objects
    obj.xAxis = 98;
    obj.yAxis = 147;
    obj.size = 37;
    objectInfos.push_back(obj);

}
```

```
25
26  }
27
28  void objectTracker::setParams(float minThreshold, float maxThreshold, bool filte
29      float minArea, bool filterByCircularity, float minCircularity, bool filterBy
30      float minConvexity, bool filterByInertia, float minInertiaRatio)
31  {
32      this->params.minThreshold = minThreshold;
33      this->params.maxThreshold = maxThreshold;
34
35      this->params.filterByArea = filterByArea;
36      this->params.minArea = minArea;
37
38      this->params.filterByCircularity = filterByCircularity;
39      this->params.minCircularity = minCircularity;
40
41      this->params.filterByConvexity = filterByConvexity;
42      this->params.minConvexity = minConvexity;
43
44      this->params.filterByInertia = filterByInertia;
45      this->params.minInertiaRatio = minInertiaRatio;
46
47  }
```

```cpp
void objectTracker::toDetect(cv::Mat* result)
{

    cv::Mat im_with_keypoints;
    cv::SimpleBlobDetector detector(this->params);
    detector.detect(*result, keypoints);
    static int s = 0, m = 0, l = 0;

    for (int i = 0; i < keypoints.size(); i++)
    {
        if (floor(keypoints[i].pt.x) == objectInfos.at(0).xAxis && floor(keypoints[i].pt.y) == objectInfos.at(0).yAxis && floor(keypoints[i].size) == objectInfos.at(0).size)
        {
            s++;
        }
        else if (floor(keypoints[i].pt.x) == objectInfos.at(1).xAxis && floor(keypoints[i].pt.y) == objectInfos.at(1).yAxis && floor(keypoints[i].size) == objectInfos.at(1).size)
        {
            m++;
        }
        else if (floor(keypoints[i].pt.x) == objectInfos.at(2).xAxis && floor(keypoints[i].pt.y) == objectInfos.at(2).yAxis && floor(keypoints[i].size) == objectInfos.at(2).size)
        {
            l++;
        }

        if (l == 5) {
            std::cout << "small objects current count : " << s << std::endl;
            std::cout << "medium objects current count : " << m << std::endl;
            std::cout << "large objects current count : " << l << std::endl;
            std::cout << "objects total count : " << s+m+l << std::endl;

        }

    }

    drawKeypoints(*result, keypoints, im_with_keypoints, cv::Scalar(0, 0, 255), cv::DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
    imshow("keypoints", im_with_keypoints);


}
```

analysisOfProductionLine

```
1  #pragma once
2  #include <string>
3  class object
4  {
5  public:
6      float xAxis;
7      float yAxis;
8      float size;
9
10 };
11
12
```