實踐篇| 機器人單目相機測距的實驗

計算機視覺戰隊 今天

以下文章來源於行走的機械人 ， 作者周旋

行走的機械人

機械設計製造及其自動化專業上進好青年，愛好技術與生活，有著大大的夢想，有個小小的情懷。讓我們在學習的路上共同進步吧！

眾志成城抗擊肺炎

如何預防新型冠狀病毒肺炎?

1 保持手衛生。咳嗽、飯前便後、接觸或者處理動物糞便後，要用流水洗手，或者使用含有酒精成分的免洗洗手液。

2 保持室內空氣流暢。避免到封閉、空氣不流通的公眾場所和人流集中的地方，必要時請佩戴口罩。

3 咳嗽和打噴嚏時用紙巾或者屈肘遮掩口鼻，防治飛沫傳播。不要打完噴嚏後揉眼睛或接觸粘膜部位。

4 良好的安全飲食習慣。處理生食和熟食的切菜板及刀具分開，做飯時徹底煮熟肉類和蛋類。

之前在做一個單目測距的小項目，大概需要就是用單目相機，對一個特定的目標進行識別並測算相機與該目標的距離。所以便去網上找了一堆教程，這裡給大家總結一下，希望給小白們一個參考。首先是基本需求：

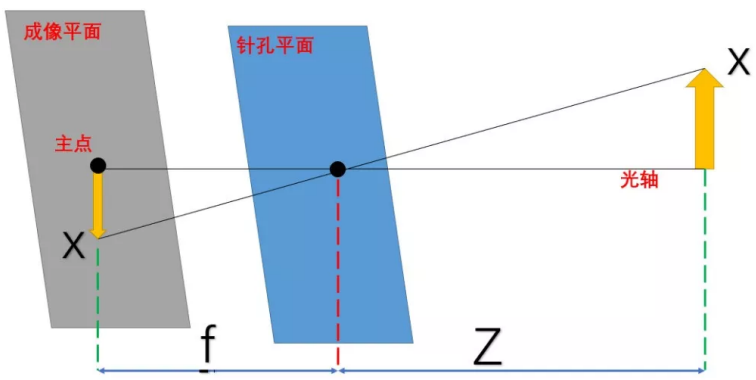
- opencv自然要會的，這咱就不多說了，會一點就行
- 需要一個攝像頭，我用的是一個畸變很大的魚眼免驅動攝像頭，大家用電腦上的那個自帶攝像頭也可以的，就是不方便。
- 需要MATLAB進行相機標定
- 需要一個編程環境，VS2017，至於VS怎麼配置opencv，可以參看一些相關的CSDN博文。

網上的方法大概有兩種，這裡主要介紹一個我身邊的大哥們都稱做**PnP問題**的一個方法，但會另外簡單介紹兩個比較簡單粗暴的，原理可行但其實效果不佳的方法。

相機畸變矯正

在用相機進行單目測距時，需要用到一個叫**相機內參**的東西，而這需要靠**相機標定**來得到。這些大概要從**相機模型**說起，相機模型是每個學opencv的同學早晚的要接觸到的吧！

我們高中都做過小孔成像的實驗，小孔相機模型就是最簡單通用的一種相機模型，這個模型我們就用下面一個圖帶過好了：



其中f為我們熟知的相機參數—— **焦距**，而光軸與成像平面的交點稱為**主點**，X表示箭頭長度，Z是相機到箭頭的距離。在上圖這個簡單且理想的小孔成像"相機"中，我們可以輕鬆的寫出黃色箭頭在現實世界坐標係與成像平面坐標系之間的轉換關係：

$$-x(世界) = f * \frac{X}{Z}$$

但是在實際相機中，成像平面就是相機感光芯片，針孔就是透鏡，然而主點卻並不再在成像平面的中心了（也就是透鏡光軸與感光芯片中心並不在一條線上了），因為在實際製作中我們是無法做到將相機裡面的成像裝置以微米級別的精度進行安裝的，因此我們需要引入兩個新的參數Cx和Cy，來對我們硬件的偏移進行矯正：

$$X_{screen} = f_x * \frac{X}{Z} + C_x \ , \ and \ Y_{screen} = f_y * \frac{Y}{Z} + C_y$$

上式中我們引入了兩個不同的焦距fx和fy，這是因為單個像素在低價成像裝置上是矩形而不是正方形。其中，fx是透鏡的物理焦距長度與成像裝置的每個單元尺寸Sx的乘積。

通過上式我們可以知道**相機內參的四個參數**了，分別是fx,fy,Cx,Cy。但在計算中，我們常通過一些數學技巧來進行一定的變換，從而得到下式：

$$\vec{q} = M * \vec{Q}$$

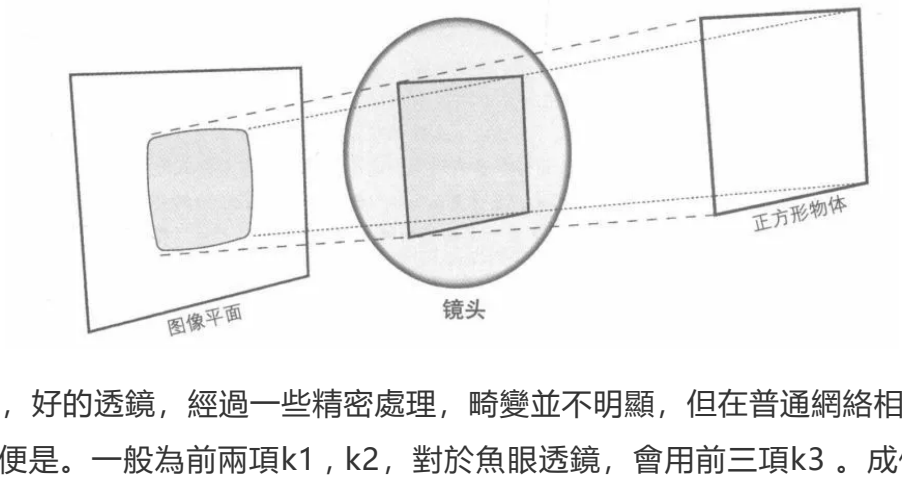
其中：

$$\vec{q} = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \ , M = \begin{bmatrix} f_x & 0 & C_y \\ 0 & f_y & C_x \\ 0 & 0 & 1 \end{bmatrix} \ , \ \vec{Q} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

通過上面的式子，我們可以將空間中的點和圖片中的點一一對應起來。式中的矩陣M就是我們常聽說的相機**內參矩陣**了。

相機外參

而有相機內參，就有相機外參了，相機外參來源於相機自身的畸變，畸變可以分為徑向畸變（有透鏡的形狀造成）和切向畸變（由整個相機自身的安裝過程造成）。

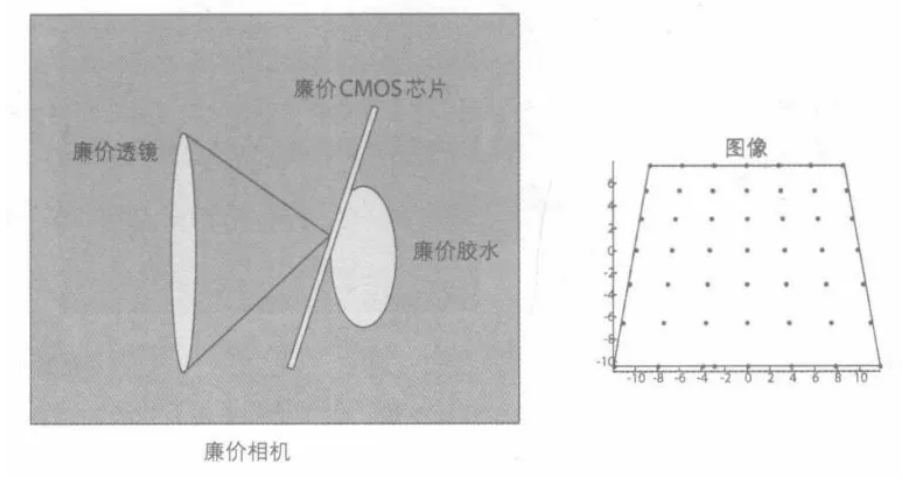


鏡像畸變是由凸透鏡本身形狀引起的，好的透鏡，經過一些精密處理，畸變並不明顯，但在普通網絡相機上畸變顯得特別突出。我們可以把畸變看作r=0附近的泰勒奇數展開的前幾項來便是。一般為前兩項k1，k2，對於魚眼透鏡，會用前三項k3 。成像裝置上某點的徑向位置可以根據以下等式進行調整，這時我們便有了3個或2個的未知變量：

$$X_{corrected} = x * (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{corrected} = y * (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

這裡（x,y）是成像裝置上畸變點的原始位置，（Xcorrected,Ycorrected）是矯正後的新位置。



切向畸變是由於製造上的缺陷使透鏡不與成像平面平行而產生的。切向畸變可以用兩個參數p1 和p2 來表示：

$$x_{corrected} = x + [2p_1xy + p_2(r^2 + 2x^2)]$$

$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2xy]$$

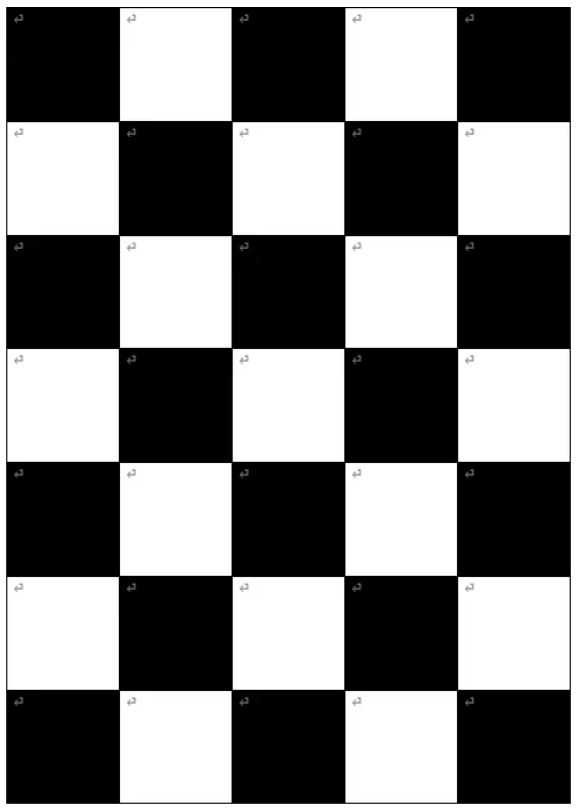
至此，我們得到了共五個參數：**K1 K2 K3 P1 P2** ，這五個參數是我們消除畸變所必須的，稱為畸變向量，也叫相機外參。

相機標定

在上文，相機內參加上相機外參一共有至少8個參數，而我們要想消除相機的畸變，就要靠相機標定來求解這8個未知參數。

說完相機模型，又要說一下相機標定了，相機標定是為了求解上面這8個參數的，那求解出這8個參數可以乾什麼呢？可以進行軟件消除畸變，也就是在得知上面8個參數後，利用上面羅列的數學計算式，將每個偏移的像素點歸位。

標定需要用到一個叫**標定板**的東西，有很多種類，但常用的大概就是棋盤圖了，棋盤要求精度需要很高，格子是正方形，買一張標定板很貴的，在csdn上下棋盤圖也要畫好多c幣，所以大家可以用word畫一張，很簡單的，只要做一個5列7行的表格，拉大到全頁，再設置每個格子的寬高來將它設為正方形再塗色就可以了。這張圖裡有符號，但打印出來就沒有了，建議大家自己畫一張就OK了。



標定過程是用MATLAB進行的，過程就不在這裡說了，CSDN上的教程一抓一大把，在完成標定後MATLAB會返回相機的內參和外參。關於原理，《學習oepncv3》這本書已經說的很好了，除了照著書抄我說不出什麼新意，但今天，原理不懂也沒有關係。

有了相機內參外參後，我們就可以進行**相機消畸變**了：

```
#include <opencv2/opencv.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

const int imageWidth = 640; //定义图片大小，即摄像头的分辨率
const int imageHeight = 480;
Size imageSize = Size(imageWidth, imageHeight);
Mat mapx, mapy;
// 相机内参
Mat cameraMatrix = (Mat_<double>(3, 3) << 273.4985, 0, 321.2298,
0, 273.3338, 239.7912,

```

```
0, 0, 1);
// 相机外参
Mat distCoeff = (Mat_<double>(1, 4) << -0.3551, 0.1386,0, 0);
Mat R = Mat::eye(3, 3, CV_32F);

VideoCapture cap1; //打开摄像头

void img_init(void)  //初始化摄像头
{
  cap1.set(CAP_PROP_FOURCC, 'GPJM');
  cap1.set(CAP_PROP_FRAME_WIDTH, imageWidth);
  cap1.set(CAP_PROP_FRAME_HEIGHT, imageHeight);
}

int main()
{
  initUndistortRectifyMap(cameraMatrix, distCoeff, R, cameraMatrix, imageSize, CV_32FC1, mapx, mapy);
  Mat frame;
  img_init();
while (1)
  {
    cap1>>frame;
    imshow("原鱼眼摄像头图像",frame);
    remap(frame,frame,mapx,mapy, INTER_LINEAR);
    imshow("消畸变后",frame);
    waitKey(30);
  }
return 0;
}
```

上面源碼中我們在32行和39行有兩個函數，就是opencv提供給我們進行消畸變的函數。

使用cv::initUndistortRecitifyMap（）函數計算矯正映射，函數原型如下：

```
initUndistortRectifyMap(
InputArray  cameraMaxtrix,     3*3内参矩阵
InputArray  distCoeffs,        畸变系数1*4向量
InputArray  R,            可以使用或者设置为noArray()。是一个旋转矩阵，将在矫正前
预先使用，来补偿相机相对于相机所处的全局坐标系的旋转。
InputArray  newCameraMatrix,  单目成像时一般不会使用它
Size    size,             输出映射的尺寸，对应于用来矫正的图像的尺寸
int      m1type,          最终的映射类型，可能只为CV_32FC1   32_16SC2，对应于map1的表示类型
OutputArray  map1,
OutputArray  map2
);
```

我們只需在程序開頭使用該函數計算一次矯正映射，就可以使用cv::remap（）函數將該矯正應用到視頻每一幀圖像。

PnP方法測距

好了到此我們對相機的那點事兒有了一點點的了解了，那什麼是PnP問題呢？在有些情況下我們已經知道了相機的內在參數，因此只需要計算正在觀察的對象的位置，這種情況下與一般的相機標定明顯不同，但有相通之處。這種操作就叫N點透視（Perspective N-Point）或PnP問題。

```
bool cv::solvePnP(
  cv::InputArray  objectPoints,     //三维点坐标矩阵,至少四个（世界坐标系）
  cv::InputArray  imagePoints,      //该四个点在图像中的像素坐标
  cv::InputArray  cameraMatrix,     //相机内参矩阵（9*9）
  cv::InputArray  distCoeffs,       //相机外参矩阵（1*4）或（1*5）
  cv::OutputArray rvec,             //输出旋转矩阵
  cv::OutputArray tvec,             //输出平移矩阵
  bool        useExtrinsicGuess = false,
  int         flags = cv::SOLVEPNP_ITERATIVE
);
```

首先來解釋一下該函數的輸出：

- **旋轉矩陣**就是一個3*1的向量，該矩陣可以表示相機相對於世界坐標系XYZ軸的3個旋轉角度。
- **平移矩陣**也是一個3維向量，可以表示相機相對於物體的XYZ軸的偏移，而這個矩陣就是我們需要求的：我們知道了相機相對於物體的位置，也就得到了距離，從而實現了測距的目的。

輸入的參數：

- **第一個參數**，是物體任意四個點在世界坐標系的三位點坐標，為什麼是四個其實很好理解，我們需要求解的是一個旋轉矩陣和XYZ軸偏移量，一共四個未知量，需要至少列四個式子才可以求解。
- **第二個參數**，我們在第一個參數中任意找的物體上的四個點在圖像中的像素坐標。
- 現在就很清楚明白了吧？通過旋轉向量和平移向量就可以得到相機坐標系相對於世界坐標系的旋轉參數與平移情況。

不過我們還要解決一個問題，如何確保這四個點的位置呢？就是，例如物體是一個正方形板子，板子長為2L，我可以選板子中心作為世界坐標系的中心，那麼我可以得到板子四個角上的坐標分別為(L,L),(L,-L),(-L,L),(-L,-L)。但如何確定圖像上哪四個點是板子的四個角呢？你就需要把板子識別出來。但如果不是個板子是個人呢？你怎麼把人分出來？這就需要更複雜的東西了，什麼語義分割啊分類器啊啥的，這裡就不多說了。

那我不取板子的四個角，利用角點檢測任意取四個點也可以，這就解決了世界坐標係與像素坐標系之間的對應問題，但又有一個新問題，如何確保這四個角點是物體身上的而不是背景上的呢？

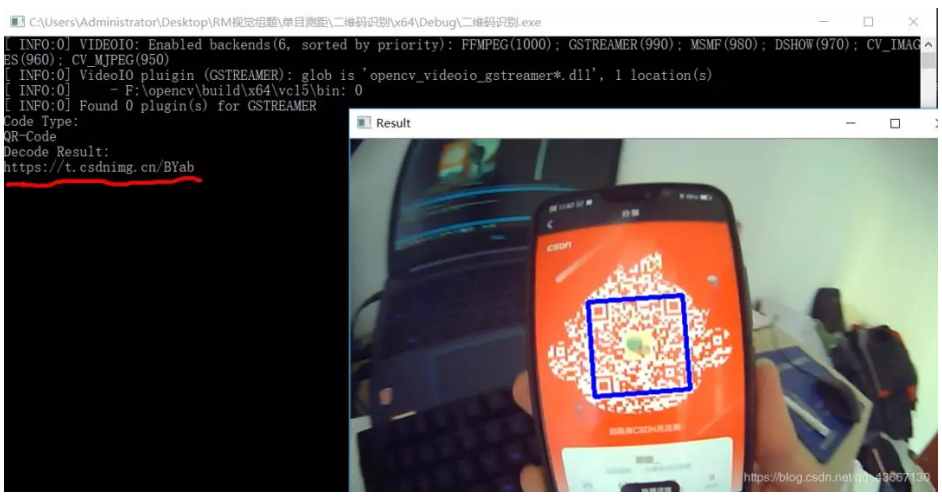
所以說這麼多，我們便引入了二維碼，我們可以直接識別二維碼來測距，這兒就要用到一個叫**ZBar庫**的東西了，它是一個可以識別二維碼或條形碼的函數庫，具體的自行百度吧。那我們還需要學一個新庫？opencv庫都還沒學明白呢，又要學一個識別二維碼的？其實不需要，這個庫的兩個例程已經可以滿足我們的需要了：

例程一：

```
#include <zbar.h>
#include <opencv2\opencv.hpp>
#include <iostream>

int main(int argc, char*argv[])
{
  zbar::ImageScanner scanner;
  scanner.set_config(zbar::ZBAR_NONE, zbar::ZBAR_CFG_ENABLE, 1);
  cv::VideoCapture capture;
  capture.open(0);  //打开摄像头
  cv::Mat image;
  cv::Mat imageGray;
std::vector<cv::Point2f> obj_location;
bool flag = true;

if (!capture.isOpened())
  {
std::cout << "cannot open cam!" << std::endl;
  }
else
  {
while (flag)
    {
      capture >> image;
      cv::cvtColor(image, imageGray, CV_RGB2GRAY);
int width = imageGray.cols;
```

```
int height = imageGray.rows;
      uchar *raw = (uchar *)imageGray.data;
      zbar::Image imageZbar(width, height, "Y800", raw, width * height);
      scanner.scan(imageZbar);  //扫描条码
      zbar::Image::SymbolIterator symbol = imageZbar.symbol_begin();
if (imageZbar.symbol_begin() != imageZbar.symbol_end())  //如果扫描到二维码
      {
        flag = false;
//解析二维码
for (int i = 0; i < symbol->get_location_size(); i++)
        {
          obj_location.push_back(cv::Point(symbol->get_location_x(i), symbol->get_location_y(i)));
        }
for (int i = 0; i < obj_location.size(); i++)
        {
          cv::line(image, obj_location[i], obj_location[(i + 1) % obj_location.size()], cv::Scalar(255, 0, 0), 3)
        }
for (; symbol != imageZbar.symbol_end(); ++symbol)
        {
std::cout << "Code Type: " << std::endl << symbol->get_type_name() << std::endl; //获取条码类型
std::cout << "Decode Result: " << std::endl << symbol->get_data() << std::endl;  //解码
        }
        imageZbar.set_data(NULL, 0);
      }
      cv::imshow("Result", image);
      cv::waitKey(50);
    }
    cv::waitKey();
  }
return 0;
}
```

這個函數可以實現打開攝像頭，並識別看到的二維碼，進而打印二維碼的類型和內容：



例程二：

```
#include <opencv2/opencv.hpp>
#include <zbar.h>

using namespace cv;
using namespace std;
using namespace zbar;

typedef struct
{
string type;
string data;
vector <Point> location;
} decodedObject;
```

```
// Find and decode barcodes and QR codes
void decode(Mat &im, vector<decodedObject>&decodedObjects)
{

// Create zbar scanner
  ImageScanner scanner;

// Configure scanner
  scanner.set_config(ZBAR_NONE, ZBAR_CFG_ENABLE, 1);

// Convert image to grayscale
  Mat imGray;
  cvtColor(im, imGray,COLOR_BGR2GRAY);

// Wrap image data in a zbar image
Image image(im.cols, im.rows, "Y800", (uchar *)imGray.data, im.cols * im.rows);

// Scan the image for barcodes and QRCodes
int n = scanner.scan(image);

// Print results
for(Image::SymbolIterator symbol = image.symbol_begin(); symbol != image.symbol_end(); ++symbol)
  {
    decodedObject obj;

    obj.type = symbol->get_type_name();
    obj.data = symbol->get_data();

// Print type and data
cout << "Type : " << obj.type << endl;
cout << "Data : " << obj.data << endl << endl;

// Obtain location
for(int i = 0; i< symbol->get_location_size(); i++)
    {
      obj.location.push_back(Point(symbol->get_location_x(i),symbol->get_location_y(i)));
    }

    decodedObjects.push_back(obj);
  }
}

// Display barcode and QR code location
void display(Mat &im, vector<decodedObject>&decodedObjects)
{
// Loop over all decoded objects
for(int i = 0; i < decodedObjects.size(); i++)
  {
vector<Point> points = decodedObjects[i].location;
vector<Point> hull;

// If the points do not form a quad, find convex hull
if(points.size() > 4)
      convexHull(points, hull);
else
      hull = points;

// Number of points in the convex hull
int n = hull.size();
```

```
for(int j = 0; j < n; j++)
    {
      line(im, hull[j], hull[ (j+1) % n], Scalar(255,0,0), 3);
    }

  }

// Display results
  imshow("Results", im);
  waitKey(0);

}

int main(int argc, char* argv[])
{

// Read image
  Mat im = imread("zbar-test.jpg");

// Variable for decoded objects
vector<decodedObject> decodedObjects;

// Find and decode barcodes and QR codes
  decode(im, decodedObjects);

// Display location
  display(im, decodedObjects);

return EXIT_SUCCESS;
}
```

該例程可以在實現例程一的功能的基礎上，還可以識別出二維碼的位置。

代碼實現

下面，如何實現測距代碼編寫呢？我們需要在上面例程二這個代碼的基礎上，加上相機畸變矯正的代碼，還要加上一段PnP函數求解的代碼：

```
vector<Point3f> obj = vector<Point3f>{
        cv::Point3f(-HALF_LENGTH, -HALF_LENGTH, 0),  //tl
        cv::Point3f(HALF_LENGTH, -HALF_LENGTH, 0),  //tr
        cv::Point3f(HALF_LENGTH, HALF_LENGTH, 0),  //br
        cv::Point3f(-HALF_LENGTH, HALF_LENGTH, 0)  //bl
    };   //自定义二维码四个点坐标
    cv::Mat rVec = cv::Mat::zeros(3, 1, CV_64FC1);//init rvec
    cv::Mat tVec = cv::Mat::zeros(3, 1, CV_64FC1);//init tvec
    solvePnP(obj, pnts, cameraMatrix, distCoeff, rVec, tVec, false, SOLVEPNP_ITERATIVE);
```

把上面三個部分融合在一起，就可以寫出我們的單目測距代碼啦：

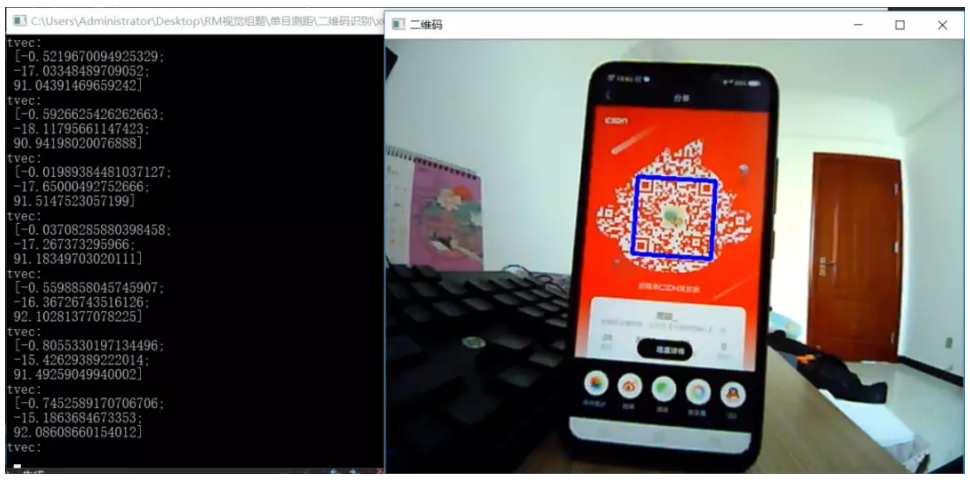
```
#include "pch.h"
#include <iostream>
#include <opencv2/opencv.hpp>
#include <zbar.h>

using namespace cv;
```

```
using namespace std;

#define HALF_LENGTH 15   //二维码宽度的二分之一

const int imageWidth = 640; //设置图片大小，即摄像头的分辨率
const int imageHeight = 480;
Size imageSize = Size(imageWidth, imageHeight);
Mat mapx, mapy;
// 相机内参
Mat cameraMatrix = (Mat_<double>(3, 3) << 273.4985, 0, 321.2298,
0, 273.3338, 239.7912,
0, 0, 1);
// 相机外参
Mat distCoeff = (Mat_<double>(1, 4) << -0.3551, 0.1386, 0, 0);
Mat R = Mat::eye(3, 3, CV_32F);

VideoCapture cap1;

typedef struct   //定义一个二维码对象的结构体
{
  string type;
  string data;
  vector <Point> location;
} decodedObject;

void img_init(void);
void decode(Mat &im, vector<decodedObject>&decodedObjects);
void display(Mat &im, vector<decodedObject>&decodedObjects);


int main(int argc, char* argv[])
{
  initUndistortRectifyMap(cameraMatrix, distCoeff, R, cameraMatrix, imageSize, CV_32FC1, mapx, mapy);
  img_init();
  namedWindow("yuantu", WINDOW_AUTOSIZE);
  Mat im;

while (waitKey(1) != 'q') {
    cap1 >> im;
if (im.empty())  break;
    remap(im, im, mapx, mapy, INTER_LINEAR);//畸变矫正
    imshow("yuantu", im);

// 已解码对象的变量
    vector<decodedObject> decodedObjects;

// 找到并解码条形码和二维码
    decode(im, decodedObjects);

// 显示位置
    display(im, decodedObjects);
//vector<Point> points_xy = decodedObjects[0].location;  //假设图中就一个二维码对象，将二维码四角位置取出
    imshow("二维码", im);

    waitKey(30);
  }

return EXIT_SUCCESS;
}

void img_init(void)
```

```
{
//初始化摄像头
  cap1.open(0);
  cap1.set(CAP_PROP_FOURCC, 'GPJM');
  cap1.set(CAP_PROP_FRAME_WIDTH, imageWidth);
  cap1.set(CAP_PROP_FRAME_HEIGHT, imageHeight);
}
// 找到并解码条形码和二维码
//输入为图像
//返回为找到的条形码对象
void decode(Mat &im, vector<decodedObject>&decodedObjects)
{

// 创建zbar扫描仪
  zbar::ImageScanner scanner;

// 配置扫描仪
  scanner.set_config(zbar::ZBAR_NONE, zbar::ZBAR_CFG_ENABLE, 1);

// 转换图像为灰度图灰度
  Mat imGray;
  cvtColor(im, imGray, COLOR_BGR2GRAY);

// 将图像数据包 装在zbar图像中
//可以参考：https://blog.csdn.net/bbdxf/article/details/79356259
  zbar::Image image(im.cols, im.rows, "Y800", (uchar *)imGray.data, im.cols * im.rows);

// Scan the image for barcodes and QRCodes
//扫描图像中的条形码和qr码
  int n = scanner.scan(image);

// Print results
for (zbar::Image::SymbolIterator symbol = image.symbol_begin(); symbol != image.symbol_end(); ++symbol)
  {
    decodedObject obj;

    obj.type = symbol->get_type_name();
    obj.data = symbol->get_data();

// Print type and data
//打印
//cout << "Type : " << obj.type << endl;
//cout << "Data : " << obj.data << endl << endl;

// Obtain location
//获取位置
for (int i = 0; i < symbol->get_location_size(); i++)
    {
      obj.location.push_back(Point(symbol->get_location_x(i), symbol->get_location_y(i)));
    }

    decodedObjects.push_back(obj);
  }
}
// 显示位置
void display(Mat &im, vector<decodedObject>&decodedObjects)
{
// Loop over all decoded objects
//循环所有解码对象
for (int i = 0; i < decodedObjects.size(); i++)

```

```
  {
    vector<Point> points = decodedObjects[i].location;
    vector<Point> hull;

// If the points do not form a quad, find convex hull
//如果这些点没有形成一个四边形，找到凸包
if (points.size() > 4)
      convexHull(points, hull);
else
      hull = points;
    vector<Point2f> pnts;
// Number of points in the convex hull
//凸包中的点数
    int n = hull.size();

for (int j = 0; j < n; j++)
    {
      line(im, hull[j], hull[(j + 1) % n], Scalar(255, 0, 0), 3);
      pnts.push_back(Point2f(hull[j].x, hull[j].y));
    }
    vector<Point3f> obj = vector<Point3f>{
        cv::Point3f(-HALF_LENGTH, -HALF_LENGTH, 0),  //tl
        cv::Point3f(HALF_LENGTH, -HALF_LENGTH, 0),  //tr
        cv::Point3f(HALF_LENGTH, HALF_LENGTH, 0),  //br
        cv::Point3f(-HALF_LENGTH, HALF_LENGTH, 0)  //bl
    };   //自定义二维码四个点坐标
    cv::Mat rVec = cv::Mat::zeros(3, 1, CV_64FC1);//init rvec
    cv::Mat tVec = cv::Mat::zeros(3, 1, CV_64FC1);//init tvec
    solvePnP(obj, pnts, cameraMatrix, distCoeff, rVec, tVec, false, SOLVEPNP_ITERATIVE);
    cout << "tvec:\n " << tVec << endl;
  }
}
```

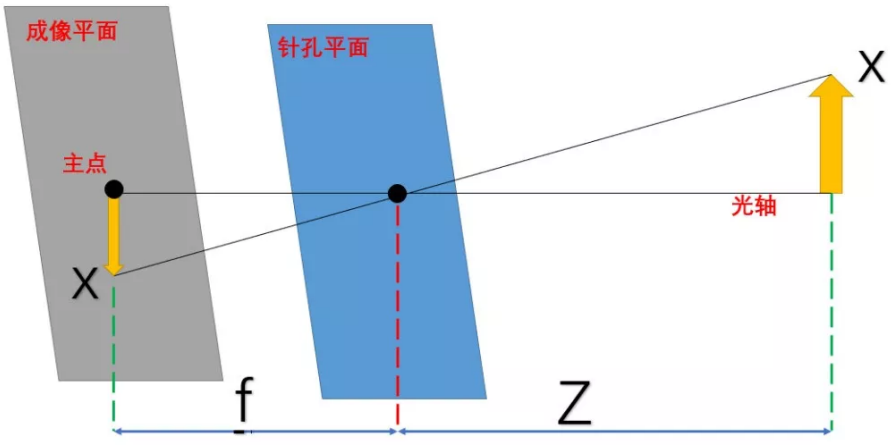
下圖是運行結果：



三個數分別是X,Y,Z的距離了，單位cm，精度可以達到0.1cm。

三角測距法

還記得文章開頭的那個小孔相機模型嗎？



三角測距法就是基於這個理想的，簡單的模型，進行的，在知道物體大小，透鏡焦距F，並測出圖像中的物體長度後，就可以基於下面公式進行計算長度Z了。

$$-X_{screen} = f * \frac{X}{Z}$$

像素塊測距法

這個方法是玩openmv時知道的，openmv封裝的單目測距算法，就是將目標對像先在固定的距離（10cm）拍一張照片，測出照片中該物體的像素面積。得到一個比例係數K，然後將物體挪到任意位置，就可以根據像素面積估算距離了。

不過這兩種方法肯定魯棒性都不咋樣。今天就到這裡啦。