**Problem6**

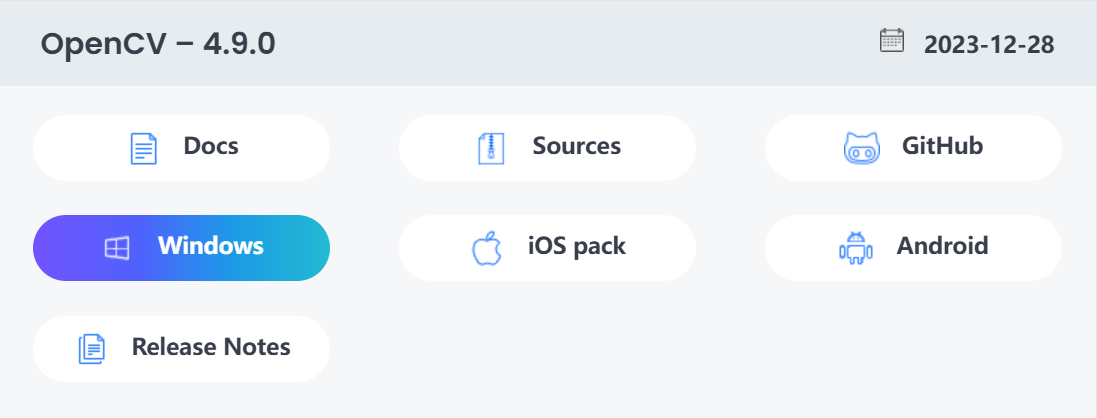
**1.Description**

ORB feature point detection and matching algorithms have been fully implemented in the OpenCV library. Please write a C++ program that invokes the OpenCV library’s algorithm for ORB feature point detection and matching for two given images, and output feature point matching results similar to the following given example.

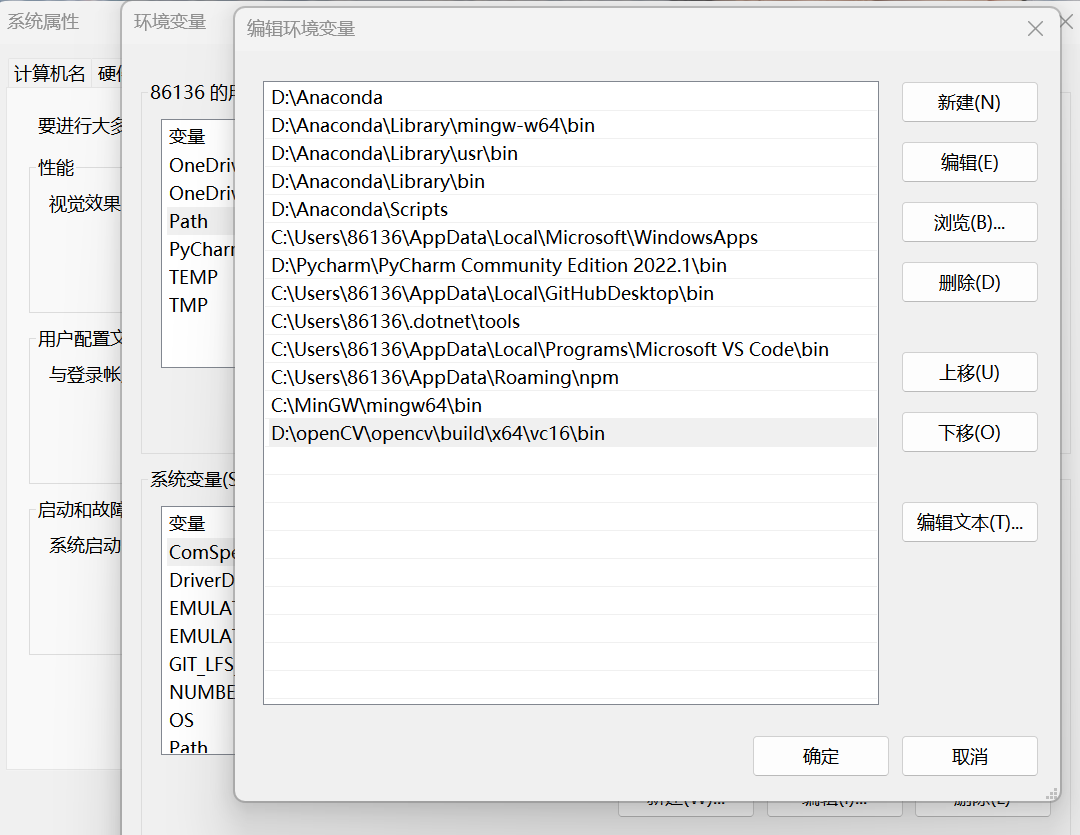
**2.Environment Setup**

**·Download OpenCV**

Visit the OpenCV website, select the appropriate OpenCV version to download (the version used in this project is OpenCV - 4.9.0), install, and remember your installation path.

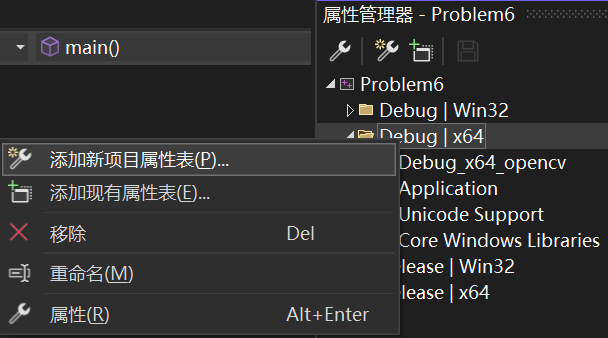


**·Add environment variables**

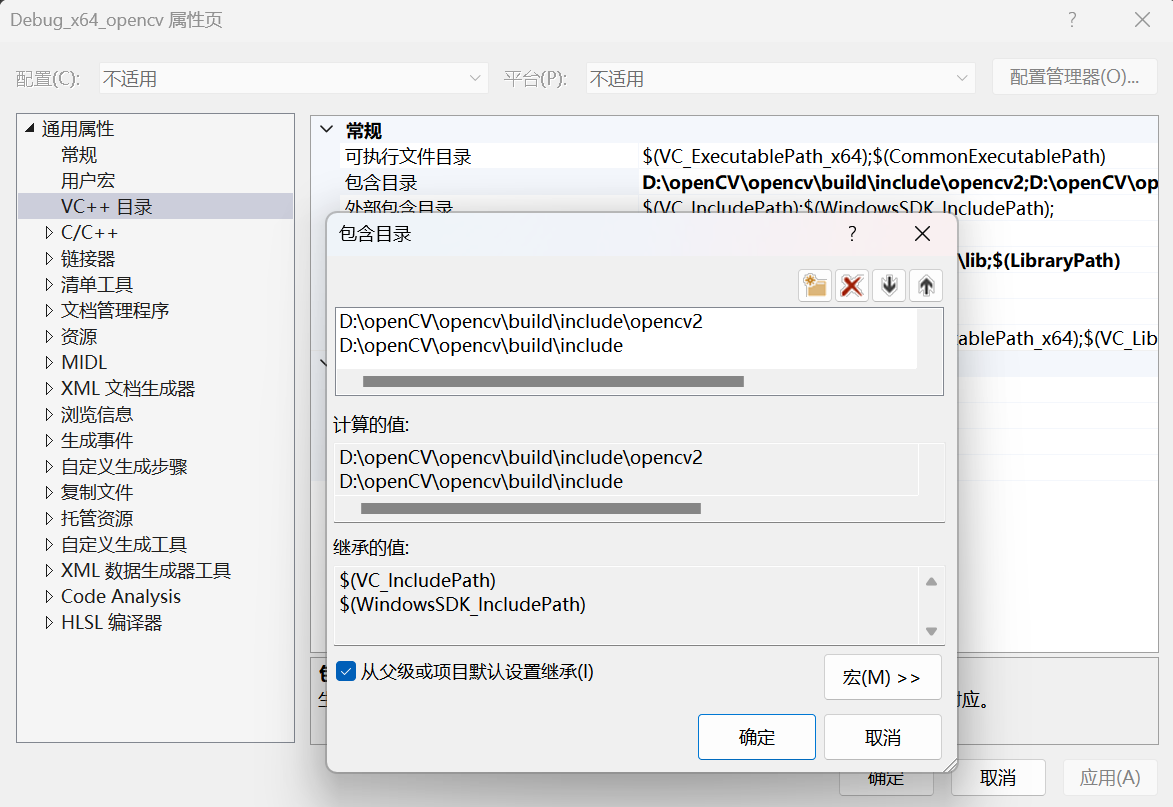
Open the system settings of the computer, select Advanced> Environment Variable> path> New, Enter the path of the folder shown in the following figure and click OK.

**·Add the project property table**

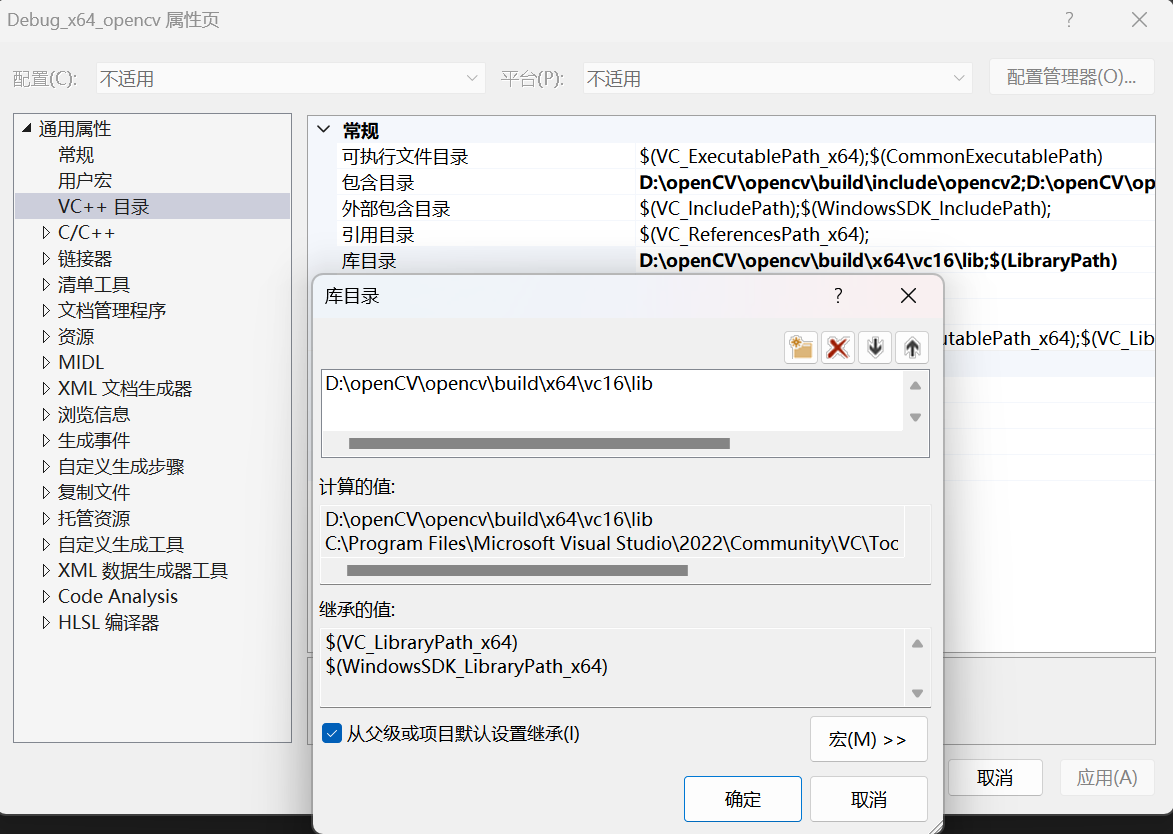
Open the project, right-click in Property Manager-DeBug | x64-add a new project property table. Then set the name such as: Debug\_x64\_opencv.props and save it.



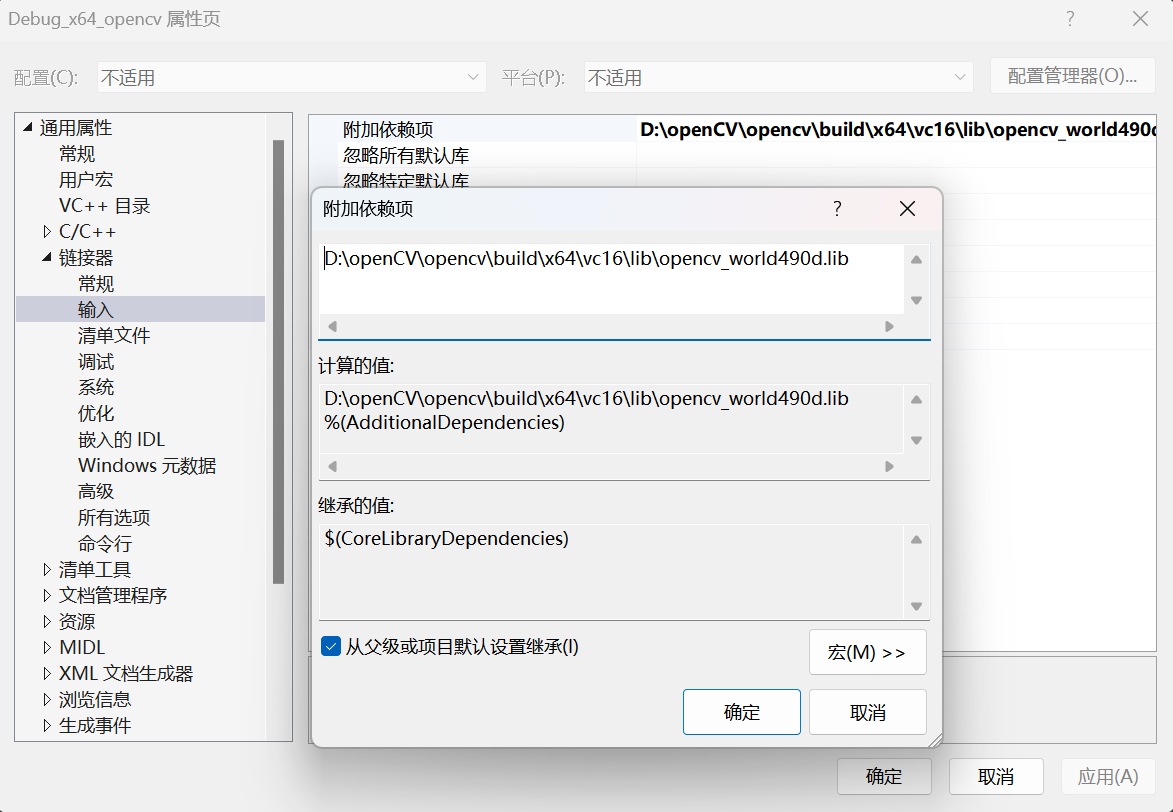
**·Configuring attribute tables**

Right-click the property table just added, and click Properties. Add the following two paths to the General Properties-VC + + Directory-General-Including Directory:

Add the following path to the General Properties-VC + + Directory-General-Library Directory**:**



Add the following path to General Properties-Linker-Enter-Additional Dependencies:



Note that the path configured above will vary with the change in where you install OpenCV and the version of OpenCV you downloaded, please note the modification.

At this point, the project OpenCV is fully configured.

**3.Solution**

ORB is a feature detection and description algorithm commonly used in computer vision, with rotation invariant and scale invariance. It combines the advantages of FAST corner detector and BRIEF feature descriptors to effectively detect keypoints in the image and generate descriptors for each keypoint.

**·Grayscale diagram conversion**

By converting two color images into grayscale maps, this step helps to improve the efficiency and stability of feature point detection.

// 将图像转化为灰度图像

cv::Mat grayImgL, grayImgR;

cv::cvtColor(imgl, grayImgL, cv::COLOR\_BGR2GRAY);

cv::cvtColor(imgr, grayImgR, cv::COLOR\_BGR2GRAY);

**·Histogram equalization**

The transformed gray-scale images were subjected to histogram equalization. This can enhance the contrast of the image, and thus improve the detection effect of the feature points.

// 直方图均衡化

cv::equalizeHist(grayImgL, grayImgL);

cv::equalizeHist(grayImgR, grayImgR);

**·ORB feature detection and descriptor computation**

Create an ORB feature detector that detects the key points in the image and calculates the descriptors for each key point. ORB is a rotation-invariant feature detection and description algorithm suitable for processing both scale and rotation-varying images.

// 创建ORB检测器

auto orbDetector = cv::ORB::create();

// 左右图像的关键点

vector<cv::KeyPoint> kpsl, kpsr;

// 关键点描述符

cv::Mat dcpsl, dcpsr;

// 检测并计算关键点和描述符

orbDetector->detectAndCompute(grayImgL, cv::Mat(), kpsl, dcpsl);

orbDetector->detectAndCompute(grayImgR, cv::Mat(), kpsr, dcpsr);

**·Match descriptors**

The descriptors of the left and right images were matched using a violent Matcher. The violence Matcher compares the distance between each pair of descriptors and matches by similarity. Match results are stored in the matchers.

// 使用暴力匹配器进行匹配

cv::Ptr<cv::DescriptorMatcher> matcher = cv::DescriptorMatcher::create(cv::DescriptorMatcher::BRUTEFORCE);

std::vector<cv::DMatch> matchers;

matcher->match(dcpsl, dcpsr, matchers);

**·Matching filtering**

The threshold matchThreshold of the matching distance was set, and only pairs of feature points with a matching distance less than that threshold were considered as valid matches. Traating through all matching pairs and mark the valid matching points as hits.

for (const auto& match : matchers){

if (match.distance < matchThreshold) { // 只处理匹配距离小于阈值的点

matchedL[match.queryIdx] = true; // 左图像的关键点被匹配

matchedR[match.trainIdx] = true; // 右图像的关键点被匹配

cv::Scalar randomColor(colorDist(gen), colorDist(gen), colorDist(gen));

cv::line(imgMatches, kpsl[match.queryIdx].pt, kpsr[match.trainIdx].pt + cv::Point2f(imgl.cols, 0), randomColor, 9);

}

}

**·image mosaicking**

The left and right images were horizontally stitched together using cv:: hconcat to form a composite image imgMatches containing two images to facilitate the presentation of the matching results in the same image.

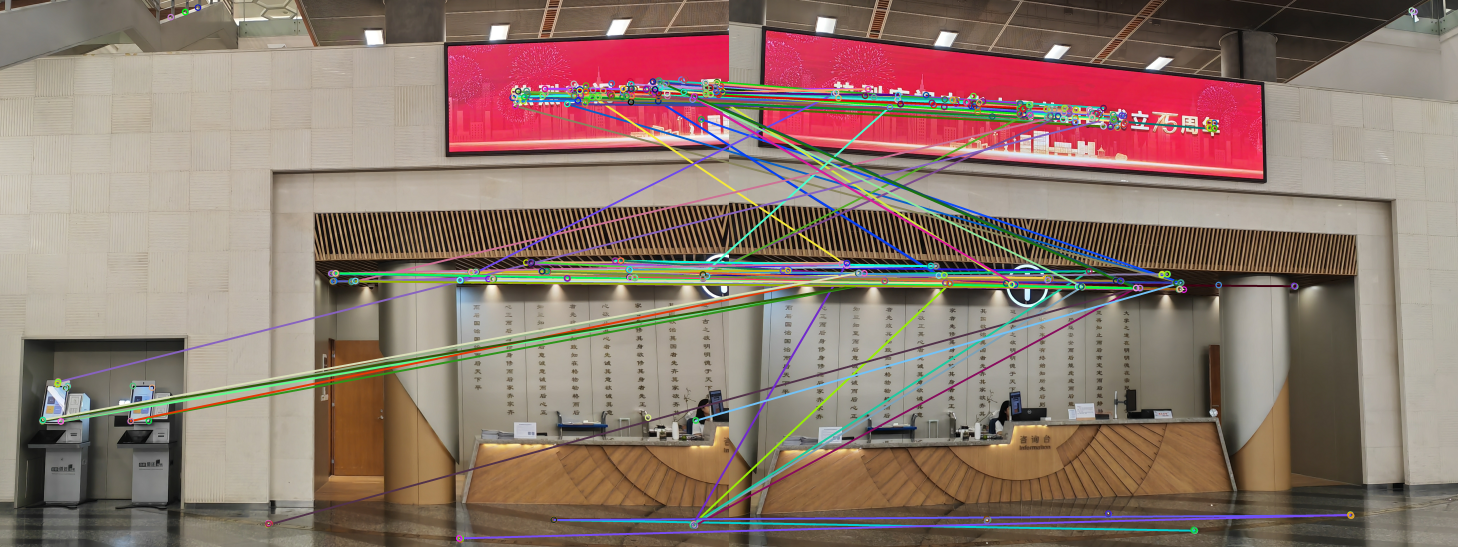
// 进行图像拼接

cv::Mat imgMatches = imgl.clone();

cv::hconcat(imgl, imgr, imgMatches);

**4.Result**

**Input:**  left.png right.png

**Output: ** Match.png