```
#include "stdafx.h"
#include "funtions.h"
#include "parameters.h"


int templateMatching_demo()
{
bool useCamera = USE_CAMERA;
std::string videoPath = VIDEO_PATH;

VideoCapture cap=createInput(useCamera,videoPath);

if (!cap.isOpened())
{
std:: cout << "fail to open video...\n" << std::endl;
return -1;
}

Mat frame;
Mat tempMat;
Mat resultMat;
Mat refMat;
Mat dispMat;

TemplateMatchModes

int cnt = 0;
while (1) {

cap >> frame;
if (frame.empty())break;

if (cnt == 0) {
Rect2d r;
r = selectROI(frame, true);
tempMat = frame(r);
tempMat.copyTo(refMat);
destroyAllWindows();
}



int match_method = 0;
matchTemplate(frame,refMat,resultMat,match_method);

normalize(resultMat, resultMat, 0, 1, NORM_MINMAX, -1, Mat());

double minVal; double maxVal; Point minLoc; Point maxLoc;
Point matchLoc;

minMaxLoc(resultMat, &minVal, &maxVal, &minLoc, &maxLoc, Mat());

if (match_method == TM_SQDIFF || match_method == TM_SQDIFF_NORMED)
{
matchLoc = minLoc;
}
else
{
matchLoc = maxLoc;
}

frame.copyTo(dispMat);
rectangle(dispMat, matchLoc, Point(matchLoc.x + refMat.cols, matchLoc.y + refMat.rows), Scalar::all(0), 2, 8, 0);


cnt++;
imshow("template",refMat);
imshow("dispMat",dispMat);
waitKey(30);

}

return 0;
}

int opticalFlow_demo()
{

vector<Point2f> corners;
double qualityLevel = 0.01;
double minDistance = 10;
int blockSize = 3, gradiantSize = 3;
bool useHarrisDetector = false;
double k = 0.04;

bool useCamera = USE_CAMERA;
std::string videoPath = VIDEO_PATH;

VideoCapture cap = createInput(useCamera, videoPath);

if (!cap.isOpened())
{
std::cout << "fail to open video...\n" << std::endl;
return -1;
}

Mat frame;
Mat grayMat;
Mat dispMat;

int maxCorners = 23;
int maxTrackbar = 100;

int cnt = 0;
while (1) {

cap >> frame;
if (frame.empty())break;

frame.copyTo(dispMat);

cvtColor(frame, grayMat, COLOR_BGR2GRAY);

goodFeaturesToTrack(grayMat,
corners,
maxCorners,
qualityLevel,
minDistance,
Mat(),
blockSize,
gradiantSize,
useHarrisDetector,
k);


for (size_t i = 0; i < corners.size(); i++)
{
circle(dispMat, corners[i], 4, Scalar(255,255,255), -1, 8, 0);
}

imshow("dispMat", dispMat);
waitKey(30);

}
return 0;
}


VideoCapture createInput(bool useCamera, std::string videoPath) {

VideoCapture capVideo;

if (useCamera) {
capVideo.open(0);
}
else {
capVideo.open(videoPath);
}

return capVideo;
}
```