第五次作业

2ORB特征点

2.1 ORB特征点

```cpp
int main(int argc, char **argv)
{

    // load image
    cv::Mat first_image = cv::imread(first_file, 0);    // load grayscale image
    if(first_image.data == NULL)
    {
        perror("first_image read failed");
    }
    cv::Mat second_image = cv::imread(second_file, 0);  // load grayscale image
    if(second_image.data == NULL)
    {
        perror("second_image read failed");
    }

    // plot the image

    //cv::imshow("first image", first_image); //显示第一个图片
   // cv::imshow("second image", second_image); //显示第二个图片//cv::waitKey(0);
    //cv::waitKey(0);
    // detect FAST keypoints using threshold=40
    vector<cv::KeyPoint> keypoints;
    cv::FAST(first_image, keypoints, 40); //调用FAST函数
    cout << "keypoints: " << keypoints.size() << endl; //输出关键点个数

    // compute angle for each keypoint
    //完成函数,计数按每个特征点的角度

    computeAngle(first_image, keypoints);
    // compute ORB descriptors
    vector<DescType> descriptors;


    //完成函数,计数ORB描述子
    computeORBDesc(first_image, keypoints, descriptors);


    // plot the keypoints
    cv::Mat image_show;
    cv::drawKeypoints(first_image, keypoints, image_show, cv::Scalar::all(-1),
                      cv::DrawMatchesFlags::DRAW_RICH_KEYPOINTS); //绘制关键点

    cv::imshow("features", image_show); //显示
    cv::imwrite("feat1.png", image_show);
    cv::waitKey(0);
    /*****************计算第二个点***************************/
     //we can also match descriptors between images
     //same for the second
    vector<cv::KeyPoint> keypoints2;
    cv::FAST(second_image, keypoints2, 40);
    cout << "keypoints: " << keypoints2.size() << endl;

    // compute angle for each keypoint,计算角度
    computeAngle(second_image, keypoints2);

    // compute ORB descriptors,计数描述子
    vector<DescType> descriptors2;
    computeORBDesc(second_image, keypoints2, descriptors2);


    // find matches,寻找匹配点
    vector<cv::DMatch> matches;
    //完成匹配函数,传入描述子
    bfMatch(descriptors, descriptors2, matches);
    cout << "matches: " << matches.size() << endl;

    // plot the matches
    cv::drawMatches(first_image, keypoints, second_image, keypoints2, matches, image_show);
    cv::imshow("matches", image_show);
    cv::imwrite("matches.png", image_show);
    cv::waitKey(0);

    cout << "done." << endl;
    return 0;
}

// -------------------------------------------------------------------------------------------------- //

// compute the angle,完成旋转部分的计数
void computeAngle(const cv::Mat &image, vector<cv::KeyPoint> &keypoints)
{
    int half_patch_size = 8;

    for (auto &kp : keypoints)
    {
        // START YOUR CODE HERE (~7 lines)
        float M10=0,M01=0;
        //图像宽为image.cols ,横坐标v
        // 高为image.rows,纵坐标 u
        //cout<<"image.cols"<<image.cols<<endl;
        //cout<<"kp.pt.x"<<kp.pt.x<<endl;
        int x = cvRound(kp.pt.x) ;
        int y = cvRound(kp.pt.y) ;
        if( x - 8 < 0 || x + 7 > image.cols || y - 8 < 0 || y + 7 > image.rows )
        {
            continue;
        }
        for (int v = -half_patch_size; v <half_patch_size ; v++)
        {
            for (int u = -half_patch_size; u <half_patch_size; u++)
            {
                float d = image.ptr<uchar> (y+v)[x+u];
                M10 = M10 + v * d;
                M01 = M01 + u * d;
            }
        }
        float ang = atan2(M10,M01)*180.f/pi; //角度
        kp.angle = ang; // compute kp.angle
        // END YOUR CODE HERE
    }
    return;
}

// -------------------------------------------------------------------------------------------------- //

// compute the descriptor
void computeORBDesc(const cv::Mat &image, vector<cv::KeyPoint> &keypoints, vector<DescType> &desc)
{
    int mycout = 0;
    for (auto &kp: keypoints) //遍历关键点
    {
        //cout<<"point:"<<mycout++<<" x: "<<kp.pt.x<<" y: " << kp.pt.y<<endl;
        DescType d(256, false); //bool数组
        //计数描述子

        for (int i = 0; i < 256; i++)
        {
            // START YOUR CODE HERE (~7 lines)
            float  ang = kp.angle * pi/180.f; //角度到弧度
           // cout<<" ang=  "<<ang<<endl;

//            Eigen::Matrix<float, 2, 2> matrix_22;
//            matrix_22<<cos(ang),-sin(ang),sin(ang),cos(ang);
//            //计算p点
//            Eigen::Matrix<float, 2, 1> matrix_p1;
//            Eigen::Matrix<float, 2, 1> matrix_p2;
//            Eigen::Matrix<float, 2, 1> matrix_q1;
//            Eigen::Matrix<float, 2, 1> matrix_q2;


//            float p1x = kp.pt.x + ORB_pattern[4*i + 0];
//            float p1y = kp.pt.y + ORB_pattern[4*i + 1];
//            float q1x = kp.pt.x + ORB_pattern[4*i + 2];
//            float q1y = kp.pt.y + ORB_pattern[4*i + 3];
            float p1x = ORB_pattern[4*i + 0];
            float p1y = ORB_pattern[4*i + 1];
            float q1x = ORB_pattern[4*i + 2];
            float q1y = ORB_pattern[4*i + 3];
//            if( p1x<0 || p1y<0 || q1x<0 || q1y<0 || p1x>image.cols || q1x>image.cols || p1y>image.rows || p1y>image.rows)
//            {
//                d.clear();
//                cout<<"pq1-wrong:"<<p1x<<" "<< p1y<<" "<<q1x<<" "<<q1y<<endl;
//                break;
//            }
            float p2x = p1x*cos(ang) - p1y*sin(ang);
            float p2y = p1x*sin(ang) + p1y*cos(ang);
            float q2x = q1x*cos(ang) - q1y*sin(ang);
            float q2y = q1x*sin(ang) + q1y*cos(ang);
            if( kp.pt.x +p2x<0 || kp.pt.y +p2y<0 || kp.pt.x +q2x<0 || kp.pt.y +q2y<0 || kp.pt.x +p2x>image.cols || kp.pt.x +q2x>image.cols || kp.pt.y +p2y>image.rows ||kp.pt.y + p2y>image.rows)
            {
                d.clear();

               // cout<<"pq2-wrong:"<<p2x<<" "<< p2y<<" "<<q2x<<" "<<q2y<<endl;
                break;
            }

            unsigned int Ip = image.ptr<uchar> (cvRound(kp.pt.y +p2y))[cvRound( kp.pt.x +p2x)];
            unsigned int Iq = image.ptr<uchar> (cvRound(kp.pt.y +q2y))[cvRound( kp.pt.x +q2x)];
            //cout<<i<<"---Ip:"<<Ip<<"Iq:"<<Iq<<endl;
            if( Ip > Iq)
            {
                d[i] = 0;
            }
            else
            {
                d[i] = 1;  // if kp goes outside, set d.clear()
            }
            // END YOUR CODE HERE
        }
        //压入描述子
        desc.push_back(d);
    }

    int bad = 0;
    for (auto &d: desc)
    {
        if (d.empty())
        {
            bad++;
        }
    }
    cout << "bad/total: " << bad << "/" << desc.size() << endl;
    return;
}

// brute-force matching,暴力匹配
void bfMatch(const vector<DescType> &desc1, const vector<DescType> &desc2, vector<cv::DMatch> &matches)
{
    cout<<"start bfMatch"<<endl;
    int d_max = 50;
    // START YOUR CODE HERE (~12 lines)
    //DescType放描述子 ,每个描述子是256的数组
    // find matches between desc1 and desc2.
    for(int i = 0; i < desc1.size() ; i++) //依次取第一个特征
    {
        //cout<<"i:"<<i<<endl;
        if( desc1[i].empty())
        {
            continue;
        }
        cv::DMatch m;
        m.distance = 256.f;
        for(auto j = 0; j < desc2.size() ;j++) //依次取第二个特征
        {

            int iDistance = 0;
            for(int k=0; k<256 ; k++)
            {
                if(desc1[i][k] != desc2[j][k])
                {
                    iDistance++;
                }
            }
            //取最小值作为匹配点
            if( iDistance <= m.distance )
            {
                m.distance = iDistance;
                m.trainIdx = j;
                m.queryIdx = i;
            }
        }
        //小于 50 保留
        if(m.distance < d_max)
        {
            matches.push_back(m);
        }
    }
    // END YOUR CODE HERE
    for (auto &m: matches)
    {
        cout << m.queryIdx << ", " << m.trainIdx << ", " << m.distance << endl;
    }
    return;
}


/*
matrix_p1[0] = kp.pt.x + ORB_pattern[4*i + 0];
matrix_p1[1] = kp.pt.y + ORB_pattern[4*i + 1];
matrix_q1[0] = kp.pt.x + ORB_pattern[4*i + 2];
matrix_q1[1] = kp.pt.y + ORB_pattern[4*i + 3];
matrix_p2 = matrix_22 * matrix_p1;
x = matrix_p2[0];
y = matrix_p2[1];

unsigned int Ip = image.ptr<unsigned short> (y)[x];

//计算q点

matrix_q1[0] = kp.pt.x + ORB_pattern[4*i + 2];
matrix_q1[1] = kp.pt.y + ORB_pattern[4*i + 3];

if( matrix_q1[0] < 0 || matrix_q1[0] > image.cols || matrix_q1[1] < 0 || matrix_q1[1] > image.rows )
{
    d.clear();
    break;
}
matrix_q2 = matrix_22 * matrix_q1;
x = matrix_q2[0];
y = matrix_q2[1];
unsigned int Iq = image.ptr<uchar> (y)[x];
*/
```

1. 根据对比得出bool值，所以说是二进制匹配
2. 阈值设置过大，错误匹配则变多
3. 使用FLANN进行快速匹配

3从E 恢复R; t

```
int main(int argc, char **argv)
{

    // 给定Essential矩阵 3*3
    Matrix3d E;
    E << -0.0203618550523477, -0.4007110038118445, -0.03324074249824097,
            0.3939270778216369, -0.03506401846698079, 0.5857110303721015,
            -0.006788487241438284, -0.5815434272915686, -0.01438258684486258;

    // 待计算的R,t
    Matrix3d R;
    Vector3d t;
    // SVD and fix sigular values
    // START YOUR CODE HERE

    std::cout<<"E :\n"<<E<<std::endl;
    //A = U *E*VT
    JacobiSVD<Eigen::Matrix3d>svd(E, Eigen::ComputeFullU | Eigen::ComputeFullV );

    Matrix3d U = svd.matrixU();
    Matrix3d V = svd.matrixV();
    Matrix3d S = U.inverse() * E * V.transpose().inverse(); // S = U^-1 * E * VT * -1
    // std::cout<<"U :\n"<<U<<std::endl;
    // std::cout<<"EE :\n"<<E<<std::endl;
    // std::cout<<"V :\n"<<V<<std::endl;
    //std::cout<<"U * EE * V2T :\n"<<U * S * V.transpose()<<std::endl;
    // END YOUR CODE HERE

    // set t1, t2, R1, R2
    // START YOUR CODE HERE
    Matrix3d t_wedge1;
    Matrix3d t_wedge2;
    Matrix3d R1;
    Matrix3d R2;
    //定义旋转矩阵
    Eigen::AngleAxisd rotation_vector1 ( M_PI/2, Eigen::Vector3d ( 0,0,1 ) );
    Eigen::Matrix3d RZ1(rotation_vector1);

    Eigen::AngleAxisd rotation_vector2 ( -M_PI/2, Eigen::Vector3d ( 0,0,1 ) );
    Eigen::Matrix3d RZ2(rotation_vector1);

    t_wedge1 = U*S*RZ1*U.transpose();
    t_wedge2 = U*S*RZ2*U.transpose();
    R1 = U*S*RZ1.transpose()*V.transpose();
    R2 = U*S*RZ2.transpose()*V.transpose();
    // END YOUR CODE HERE
    cout << "R1 = " << R1 << endl;
    cout << "R2 = " << R2 << endl;
    //vee  反对称矩阵到向量
    cout << "t1 = " << Sophus::SO3::vee(t_wedge1) << endl;
    cout << "t2 = " << Sophus::SO3::vee(t_wedge2) << endl;

    // check t^R=E up to scale
    Matrix3d tR = t_wedge1 * R1;
    cout << "t^R = " << tR << endl;

    return 0;
```

4用GN实现bundle adjustment

```
int main(int argc ,char** argv)
{
    VecVector2d p2d;
    VecVector3d p3d;
    Matrix3d k; //内参
    double fx=520.9,fy=521.0,cx=325.1,cy=249.7;
    k<<fx,0,cx,0,fy,cy,0,0,1;

    //load points into p3d and p2d
    //START YOUR CODE HERE
    ifstream file1(p3d_file);
    if(!file1.is_open())
    {
        perror("p3d open failed");
    }
    ifstream file2(p2d_file);
    if(!file2.is_open())
    {
        perror("p2d open failed");
    }
    while (!file1.eof())
    {
        double p3[3] = {0};
        for (auto &p:p3)
        {
            file1 >> p;
        }
        p3d.push_back(Vector3d(p3[0], p3[1], p3[2]));
    }
    while (!file2.eof())
    {
        double p2[2]={0};
        for (auto& p:p2)
        {
            file2>>p;
        }
        Vector2d v(p2[0],p2[1]);
        p2d.push_back(v);
        //cout<<"p2d "<<v<<" p3d "<<Vector3d(p3[0],p3[1],p3[2])<<endl;
    }
    //END YOUR CODE HERE
    assert( p3d.size() == p2d.size() ); //如果二者行数相等，则而已继续执行，否则
程序停止运行

    int iterations=100;
    double cost=0,lastcost=0;
    int nPoints=p3d.size();
    cout<<"points: "<<nPoints<<endl;


    Matrix3d I = Matrix3d::Identity(); //声明单位矩阵
    Vector3d t ;
    t.setZero();
    //cout<< "I:\n"<<I<<endl;
    //cout<< "t:\n"<<t<<endl;
    Sophus::SE3 T_esti(I,t);//变换矩阵
    //Sophus::SE3 T_esti;
    cout<<"T_esti:\n"<<T_esti.matrix()<<endl;



   for (int iter = 0; iter <iterations ; ++iter) //迭代100次
    {
        Matrix<double,6,6> H = Matrix<double,6,6>::Zero();
        Vector6d b = Vector6d::Zero();

        cost=0;
        //compute cost
        for (int i = 0; i <p3d.size() ; ++i) //遍历数组
        {
            //compute cost for p3d[i] and p2d[i]
            //START YOUR CODE HERE
            Vector2d ui=p2d[i]; //2*1 p2p点
            //Vector3d pi=p3d[i];
            //（7.34）4*4 的变换矩阵 × 4*1的3d点

            Vector4d pii = T_esti.matrix()*Vector4d(p3d[i][0],p3d[i][1],p3d[i][2],1);

           // cout<< "pii:\n"<< pii <<endl;
           // cout<< "k:\n" << k << endl;
            // 3*1的s*u =3*3 的内参 × 3*1的(SE3*P)
            Vector3d pi=k*Vector3d(pii[0],pii[1],pii[2]);//s*u

            cout<< "pi:\n"<< pi<<endl;

            //求误差
            Vector2d e( ui[0]-pi[0]/pi[2] , ui[1]-pi[1]/pi[2] ); //除以pi[2]是归一化过程，误差为2*1
            cost += e(0,0)*e(0,0)+e(1,0)*e(1,0);//e.transpose()*e;
            //END YOUR CODE  HERE

            //compute jacobian
            Matrix<double,2,6> J; //定义2*6 雅克比

            //START YOUR CODE HERE
            //7.45
            J(0,0)=-fx/pii[2];
            J(0,1)=0;
            J(0,2)=fx*pii[0]/(pii[2]*pii[2]);
            J(0,3)=fx*pii[0]*pii[1]/(pii[2]*pii[2]);
            J(0,4)=-fx-fx*pii[0]*pii[0]/(pii[2]*pii[2]);
            J(0,5)=fx*pii[1]/pii[2];
            J(1,0)=0;
            J(1,1)=-fy/pii[2];
            J(1,2)=fy*pii[1]/(pii[2]*pii[2]);
            J(1,3)=fy+fy*pii[2]*pii[2]/(pii[0]*pii[0]);
            J(1,4)=-fy*pii[0]*pii[1]/(pii[2]*pii[2]);
            J(1,5)=-fy*pii[0]/pii[2];
            //END YOUR CODE HERE

            H+=J.transpose()*J;
            b+=-J.transpose()*e;
        }

        //solve dx
        Vector6d dx;

        //START YOUR CODE HERE
        dx=H.ldlt().solve(b);
        //END YOUR CODE HERE

        cout<<"iteration "<<iter<<" cost="<<cout.precision(12)<<cost<<endl;
        if(isnan(dx[0]))
        {
            cout<<"result is nan!"<<endl;
            break;
        }

        if(iter>0&&cost>=lastcost)
        {
            //cost increase,update is not good
            cout<<"cost: "<<cout.precision(12)<<cost<<", last cost: "<<cout.precision(12)<<lastcost<<endl;
            break;
        }

        //update your estimation
        //START YOUR CODE HERE
        T_esti=Sophus::SE3::exp(dx) * T_esti;

        //END YOUR CODE HERE
        lastcost=cost;
        cout<<"iteration "<<iter<<" cost="<<cout.precision(12)<<cost<<endl;
    }

    cout<<"estimated pose: \n"<<T_esti.matrix()<<endl;

    return 0;
}
```