Indirect Visual Odometry with Optical Flow

Chenguang Huang and Andong Tan

Technische Universität München, Chair of Computer Vision and Artificial Intelligence
chenguang.huang@tum.de andong.tan@tum.de

Abstract

With the increasing need of robotic application in many different domains like autonomous driving and factory cooperative robots, the navigation becomes an important topic. The camera, at the same time, is nowadays a type of cheap sensor, which is powerful in extracting useful information from visual input. Therefore, vision based navigation turns out to be an effective solution. In vision based navigation, visual odometry is an important module. This paper introduces the implementation details of visual odometry using the method of optical flow and compares it with the method without optical flow. A manual implementation of optical flow is also presented. One novel point of the method with optical flow is that we segment the image using grids and use the number of empty cells to decide key frames. The influences of related parameters are evaluated in the end.

Contents

1 Introduction

Computer vision related tasks in robotics are attracting more and more attention. One typical task is the visual odometry. It is used widely in applications which require the depth information of objects without directly relying on distance measurement sensors like Lidar. Besides, many applications use visual odometry to help with the motion planning of some specific agent. Therefore, visual odometry could cause critical problems if it is not reliable. To ensure the safety of the agent, a common choice is to use mathematically provable methods to realize the function rather than using currently unexplainable techniques like deep learning. Thus it worths looking into the implementation details of visual odometry using traditional explainable methods.

To estimate the depth of some specific object, at least two images are needed if there is no prior assumption in how the world is constructed. However, before estimating the depth through triangulation, corresponding points in at least two images which describe the same 3D point should be found. The first way to achieve this is doing key point detection in both images, and finding the matching point pairs through key point descriptor similarity comparison. The second way to achieve this is the optical flow, which computes an estimated position of a pixel in the second image according to two raw image inputs and its position in the first image. The above two ways are suitable to find the point matching pairs between consecutive frames.

This paper compares the above two methods based on a two-camera system (left and right camera).

The following sections are structured as below: Section II presents some basic concepts used in the two methods. Section III describes the pipeline of the two methods. Different strategies used in the implementation are summarized in section IV. Section V shows the implementation details of the method using optical flow as well as the manual implementation of optical flow. The results are evaluated in Section VI, and Section VII concludes the work.

2 Basic Concepts

To understand the methods compared better, basic concepts of visual odometry, optical flow, key point detection and point matching are summarized in this section.

2.1 Visual Odometry

Odometry means the estimation of the change in position over time. And visual odometry refers to the estimation of the motion of a camera in real time through sequential images. In the context of vision based navigation, where cameras are often integrated in the robot, visual odometry can be used to estimate the ego motion of the mobile robot, and thus helps to build a map in real time around the robot to support the navigation [1].

2.2 Optical Flow

The optical flow is apparent 2D motion which is observable between consecutive images. It can also be understood as pixel-wise motion estimation, because the optical flow calculates the motion of a specific pixel between two consecutive frames. Two main types of optical flow calculation are the Lukas & Kanade method (indirect method) and Horn & Schunck method (variational method) [3] [4]. These two methods have different assumptions but both calculates the optical flow through an optimization process.

In Lukas & Kanade method, it assumes that (i) the motion is constant in a local neighborhood (ii) the brightness of a specific pixel is constant in different frames. Under the above assumptions, the energy function in the Lukas & Kanade method is formulated as follows:

$$E(v) = \int_{W(x)} |\Delta I(x', t)^T v + I_t(x', t)|^2 dx' \tag{1}$$

where $I(x', t)$ denotes the brightness of position x' at time t in the image, I_t denotes the derivative of brightness with respect to the time t, and $W(x)$ denotes the neighborhood of pixel x'. The optical flow v is calculated via the minimization of the above energy function. It generates sparse flow vectors [3].

In contrast, the Horn & Schunck method assumes (i) the brightness of a specific pixel is a constant in different frames (ii) the motions are spatially smooth. This method generates dense flow vectors. The energy function of this method is:

$$E(u, v) = \int\int [(I_x u + I_y v + I_t)^2 + \alpha^2(||\Delta u||^2 + ||\Delta v||^2)] dxdy \tag{2}$$

where I_x and T_y are the derivatives of brightness with respect to x and y axis, respectively. u and v are velocity in vertical and parallel directions. In the end, optical flow is obtained through solving for u and v variables [4].

2.3 Key Point Detection

The real environment is continuous and complex. To successfully navigate in such en environment with a reasonable computation load, key information should be focused on. In the image, one kind of typical key information is the corner point. This can be detected through the anaysis of the gradient of the image [5]. Typical methods include Foerstner and Harris detector [6]. More recently, detectors like BRIEF and Shi-Tomasi are quite popular [8] [9], and have many extended versions. In our experiments, key points construct the landmarks and their corresponding observations.

2.4 Point Matching

To construct a 3D map, the matching relationship of points between frames are important, otherwise it will cause big error in the triangulation and possibly make the optimization process in determining the 3D position of a landmark unable to converge.

Finding matches between points in consecutive frames are typically done through the comparison of descriptors of two points. If the similarity between two descriptors is high enough, the two corresponding points are considered to be different projections of the same real world point in two different frames.

One method to compute the descriptor is called the ORB descriptor [7]. ORB descriptor extends the BRIEF descriptor [8] and uses a trained pattern to describe the key point. According to a set of principles based on the similarity, the correspondence of points between frames can be calculated.

3 Structure Design

Our pipeline assumes that the stereo cameras have already been calibrated, which means that intrinsic matrix and relative transformation between two camera are already known. To make the pipeline easy to understand, the pipeline is demonstrated in three different conditions: first key frame, second or later key frames and non-keyframes. The criteria of deciding whether a certain frame is a key frame will also be discussed later. A general pipeline is shown in the Figure 1.

3.1 First Key Frame

In the first key frame (often also the first frame), we firstly use Shi-Tomasi method to detect key points in the image of left camera and use optical flow to acquire the corresponding key points in the image from right camera. To increase the correctness of the result of optical flow, optical flow is performed backward. If the resulting key points are adequately close to original input key points in the left image, this pair of points is considered as valid. Moreover, to filter out more outliers, the relative transformation acquired from calibration is used to calculate the essential matrix and the epipolar constraint check is performed. In the next step, the camera is localized by initializing the first pose of left camera with identity matrix. Finally, the triangulation rule is used to calculate the 3D position of landmarks in world coordinate frame.

3.2 Second or Later Key Frames

To simplify the description of this part, we denote the image from left camera as ”left image” and image from right camera as ”right image”. For second or later key frames, firstly optical flow is used to acquire corresponding key points from last left image to current left image. Also a backward check is performed as in the first key frame. Later a grid on the current left image is made and empty cells in the grid which do not contain key points are detected. In the next step, the Shi-Tomasi method is used to detect new key points inside these empty cells. In this period, we have two kinds of key points in the left image. One type includes key points acquired with optical flow from last frame, which we will call old key points. The other type includes newly detected key points with Shi-Tomasi method in empty cells, which we will call new key points. Given these key points in the left image, we perform optical flow from left image to right image and backward check (perform optical flow back from right image to left image) to filter out noisy points. And then we use epipolar constraint to find inliers. Later on we use Random Sample Consensus (RANSAC) algorithm to localize camera with only inliers of old key points (calculated from last frame) in the left image. Finally we use triangulation to calculate new landmarks with newly detected key points and add key points calculated through optical flow to observations of old landmarks.

3.3 Non-keyframes

For non-key frames, frame to frame optical flow is performed first and the grid is made as in the second case above. But instead of detecting key points in empty cells, here only the number of empty cells is counted, which will be used as one of the criteria to determine key frames. Later camera is located with the RANSAC algorithm.

4 Strategies

This section describes the detailed strategy used in the implementation of indirect visual odometry with optical flow. As the main difference between optical flow based method and

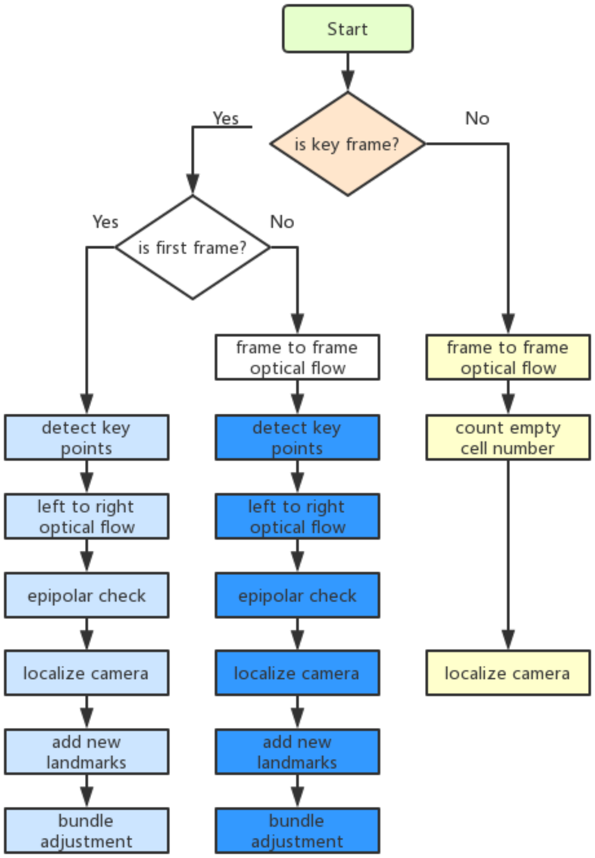


Figure 1: Pipeline structure

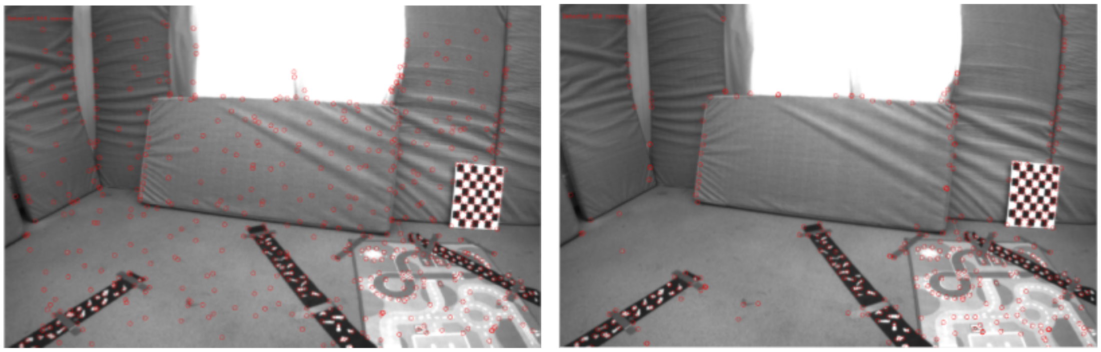
fully key point descriptor matching based method is in the key point matching process and key frame choosing process, these two aspects are highlighted in the rest of this section.

4.1 Key Framing Strategies

The criteria of key framing is of significant importance. It will affect the accuracy and efficiency of our algorithm. On one hand, if one frame can be easily regarded as a key frame, the interval between key frames will become very small. So the optimization will be limited in a small range, focusing too much on local features instead of a global view. In the meantime, more key frames means more optimization time, which makes the algorithm inefficient. On the other hand, if the key framing criterion is too strict so that only very few frames can be selected as key frames, the pipeline is more likely to collapse during the run. This is because we only detect new key points in key frames and if the interval between key frames is too big, the base line will be too large for optical flow to find correspondences through all the frames.

Once there are not enough correspondences, the camera localization will become very unstable and the algorithm will fail.

In our pipeline two criteria are used to decide whether the next frame is a key frame. The first criterion is the number of inliers. One necessary step in each time frame is the camera localization. By using RANSAC method, a certain number of inliers are selected for determining the pose of camera. To guarantee there are enough key points for localization, a threshold is set for the number of inliers. If the number of inliers is lower than the threshold, the next frame is set as a key frame to detect more points. The second criterion is the number of empty cells. As stated in above section, for every non-keyframe a grid is made on the image recorded by the left camera and the number of empty cells will be counted. A large empty cells number means the feature points calculated by optical flow gather in a small area of the image, indicating that the existing landmarks are leaving the current observation view. Once the empty cells number exceeds a threshold, the next frame is set as a key frame to enable detection of new key points. In this way, the key points in the image will be adjusted from time to time to distribute relatively uniformly, making localization more stable.



(a) detect in subimage

(b) detect in whole image

Figure 2: key points detection strategies

4.2 Key Points Detection Strategies

In every key frame, key points are detected in empty cells. We try two different implementations of key points detection in our pipeline. In the first implementation, empty cells are extracted as a series of subimages and the best one key point is detected (if at least one key point can be detected) in each sub-image separately. In the second method, key points are detected in the whole image, the best key points (max. 1500) are chosen and only key points in empty cells are kept. The difference between these two key point detection methods is shown in Figure 2. It is easy to see that the number of detected key points in the first implementation is larger than the second implementation. More importantly, the distribution of key points in the first implementation is more uniform. To quantify the influence of key point detection methods, the whole pipeline is run on euroc dataset and the root mean square error (RMSE) of camera trajectories are compared. Finally, it is found that the first implementation performs slightly better than the second implementation.

5 Implementation

5.1 Lucas and Kanade Optical Flow with OpenCV

To track the key points between different frames, three functions are implemented: *OpticalFlowBetweenFrame_opencv_version*, *OpticalFlowToRightFrame_opencv_version* and *OpticalFlowFirstStereoPair_opencv_version*. All of these three functions need to input one source image, one target image, source key points data and target key points data. For the algorithm of optical flow, we use the Lucas and Kanade method implemented in OpenCV and use patch window size (21 x 21) and pyramid level 4. A backward checking using optical flow is also performed to improve the robustness of tracking. Only when the distance between the backward result point and its corresponding input point is smaller than 1 pixel can the tracking result be regarded as valid.

Among these three functions, the difference lies in the correspondence map they establish. In *OpticalFlowBetweenFrame_opencv_version*, we establish match data that match feature ID and track ID. In this way, the key points in current frame can have correspondence with key points of existing landmarks. In *OpticalFlowToRightFrame_opencv_version*, we establish match between key points in the left and right images. Furthermore, we have to distinguish whether the optical flow result key points in the right image are from old key points or from newly detected key points. So we use two match data, one for matching between old key points in the left image and their resulting key points in the right image, one for newly detected key points in the left image and their correspondence in the right image. In *OpticalFlowFirstStereoPair_opencv_version*, we only create stereo matching data. Because this function is for stereo track in the first time step, we do not need to distinguish whether the key points are new or old.

5.2 Light Intensity Invariant Optical Flow with Manual Implementation

The variation of light intensity of object overtime might fail Lucas and Kanade optical flow. Therefore, a more robust method of optical flow should be used to increase the stability of the pipeline. We manually implement an optical flow algorithm. In this implementation, we scale the patch intensity with the mean value, using only the local relative intensity of patch to track so that the influence of intensity variation is mitigated. The idea of the algorithm is to minimize such a residual function with Gauss-Newton method [10]:

$$r_i\left(\xi\right) = \frac{I_{t+1}\left(Tx_{\ i}\right)}{\overline{I}_{\ t+1}} - \frac{I_t\left(x_{\ i}\right)}{\overline{I}_{\ t}} \forall x_{\ i} \in \Omega$$

The basic algorithm of illumination invariant optical flow is offered in appendix 1. The implementation of this function is in *project_eval* branch. The function name is *OpticalFlowLK*.

5.3 Cell and Grid

In order to make cells more conveniently, we implement a class to represent a single cell and a function to make grid on image. *Cell* is a class which contain the coordinates of four corners of a cell and also the key points number inside this cell. By using function *makeCells*, we can create a vector of *Cell*s and initialize the key points number in each cell with 0. *makeCell* needs the input of the width and height of an image, how many rows and columns of cells, and a vector used to store cells. To modify the rows number and columns number easily, we set global variables *rnum* and *cnum* at the beginning of *odometry.cpp* for tuning.

In order to determine whether next frame is a key frame, one option is to judge whether the empty cells number exceeds a certain threshold. We also implement a function, *check_num_points_in_cells*, which can update the number of key points in a vector of cells when key points and the cells vector are provided. *sparsity* function returns the empty cells number and their indices when a vector of cells is given.

5.4 Key Points Detection

We implement two version of key points detection function. They use different strategies as stated in strategy section above.

add_new_keypoints_from_empty_cells detects key points in each subimage. By inputing the cells vector, empty cells indices and image on which key points will be detected, the function extracts subimage specified by each empty cell and performs Shi-Tomasi key points detection algorithm on that subimage. There is some details we need to pay attention to. First, the detected key points coordinates are relative to the subimage, so we need to add the coordinate of top left corner of the subimage to the result to acquire the position of key points in the whole image. Second, considering that optical flow algorithm uses patch around key points for tracking, we ignore the key points detected in the boundary subimage to ensure the patches can entirely lie in the whole image. Third, the maximum key points number in each subimage is set to 1.

add_new_keypoints_from_empty_cells_v2 detects key points in whole image and then deletes the key points in non-empty cells. The maximum number of key points is also given into the function.

6 Evaluation

This section evaluates the performance difference between the matching method using optical flow and the matching method fully relying on key point detection, as well as the performance of the optical flow method under different set of hyper parameters. The evaluation focuses on three main aspects: precision, execution time, and visualization.

As described in the previous section, a grid is used to segment the image frame and decide whether the next frame is a key frame. A basic set of parameters for the grid of our experiments is as the following table 1:

Table 1: **Basic set of parameters**

Image height $h(pixel)$	Image width $w(pixel)$	Grid size $s(pixel \times pixel)$	Min. num. of key points m
480	752	32 × 32	100

Table 2: **RMSE precision under different number of Max. empty cells**

Max. number of empty cells (%)	0.35	0.4	0.42	0.44	0.46
RMSE	0.1303	0.1295	0.1099	0.1563	0.1304
Rmse using method without optical flow:	0.1012				

The grid size is designed to be so such that it's divisible by the image height and width. Besides, it is found through experiments that square grids brings higher precision in the following experiments than rectangular grids. Similarly, the minimum number of key points 100 is found through experiments. To simplify the following discussion, the grid size and the minimum number of key points are set to be a constant, and the influence of other hyper parameters are discussed in the rest of the section.

6.1 Precision

The table 2 shows how precision measured as RMSE changes when the maximum number of empty cells percentage changes. This variable is designed to control the maximum area of the image where there is no key point. When the empty area is too big, it means a lot of information from these areas is lost and the construction of the map could be not precise enough. When this happens, the next frame will be set as a key frame, and key point detection will be executed.

From the table 2 it is clear that when the maximum number of empty cells percentage is too small or too large, the precision of the built map will decrease. This means, if detection of new key points is done too frequently, the constructed map has too many key points, which means too many noises, as the key point detection itself is also not 100% precise. However, if key points are detected too sparsely, we would have too less key points to construct the map, which indicates that even some small noises could cause a big error in map construction and thus cause an increased RMSE value. In the best case, the RMSE will reach a similar value as the method without using optical flow. Figure 3 shows more experiment data in a visualized way.

6.2 Execution Time

The evaluation on execution time includes three parts: detection time, optimization time, and key point matching time. Choosing these three aspects is because: all these three steps are common in both optical flow based method and the method without optical flow, and the difference between these two methods has a big impact on the execution time of these three steps.

6.2.1 Detection Time

The table 3 shows how detection time changes with change of maximum number of empty cells percentage. Firstly, it is obvious that the detection time in the method using optical flow is much less than the method without optical flow, because the method using optical flow has replaced a lot of key point detection operation and found the matching points between frames using optical flow. Secondly, it's reasonable to see that with an increasing threshold for

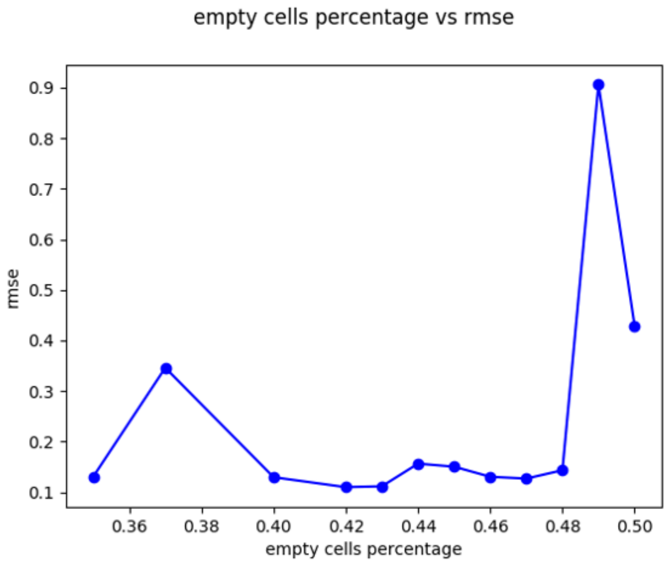


Figure 3: Visualized version of RMSE change with different parameter of max. number of empty cells percentage

Table 3: **Detection time under different number of Max. empty cells percentage**

Max. number of empty cells (%)	0.35	0.4	0.42	0.44	0.46
Detection time (s)	3.03643	2.73232	2.43286	2.62328	2.26326
Detection time using method without optical flow:	44.0214				

maximum empty cells percentage, the detection time generally decreases. This is due to the decreasing frequency of key frames, therefore the detection operation is executed less frequently. The experiment result shows that there is a slight increase on detection time when the threshold is set to 0.44 in comparison to 0.42, we think this could due to the fact that some frames have more textures and corners than the others, and a specific choice of 0.44 occasionally lets us have more key frames in these pictures with a lot of texture and corners. And it thus increases the detection time slightly.

Table 4: **Optimization time comparison**

	Method with optical flow	Method without optical flow
Opt. time (s)	305.043	65.8619
Num. landmarks	290657	221886
Num. observation	1939460	559563
Num. key frames	414	170

Table 5: **Key point matching time comparison**

Point descriptor matching time (s):	5.79
Optical flow calculation (s):	117.4 143.1

6.2.2 Optimization Time

Optimization takes the most time of the whole pipeline. Here we choose the best parameter for maximum percentage of the number of empty cells, which is 0.42 for the method with optical flo and compare it with the method without optical flow. The results are shown in table 4.

The optimization time of the optical flow method is much higher than the method fully depending on key point detection. Although the number of landmarks are in similar level, the number of key frames and observation in optical flow method is much higher than the other one, which causes the huge difference in optimization time. As the optimization process only happens in key frames, an increased number of key frames naturally increases the total optimization time. Here comes automatically the question: Why do we have more key frames and observations in optical flow based method?

Firstly, selection of key frames based on counting empty cells (optical flow method) rather than key point number (non optical flow method) in the image leads to a denser choice of key frames. Therefore the number of key frames is much higher.

Secondly, a possible reason for the large number of observations is that the pipeline forces a key point detection operation in every small grid of key frames. Although this has the advantage that the detected key points are more widely distributed in the whole image to help with a preciser optimization process, it has the disadvantage that many noises are also introduced and many noisy points are considered as an observation of some specific landmark.

The above reasons together lead to the difference in optimization time between two methods.

6.2.3 Key Point Matching Time

As a replacement of point descriptor comparison between frames, optical flow is used to track the key point between frames. Therefore, we compare the execution time of optical flow and key point matching through descriptors in the table 5.

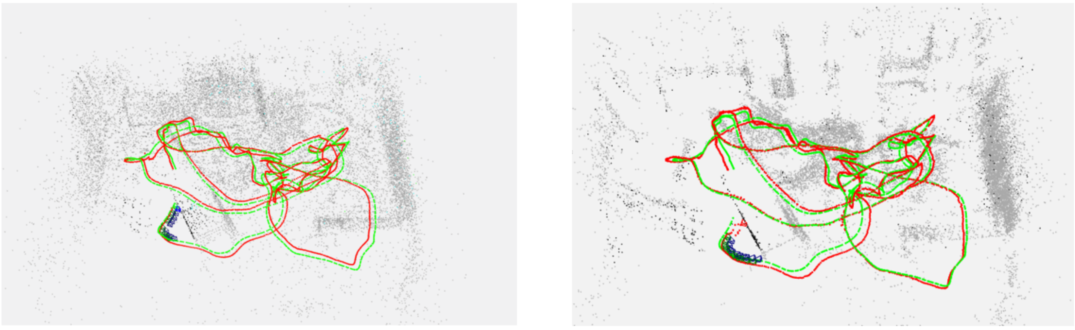
The optical flow calculation time is much higher than the key point matching using descriptors. This is due to the fact that optical flow calculation includes an optimization process and the result is calculated iteratively, but descriptor comparison is a rather direct method which only needs several basic mathematical operations. Besides, the descriptor used (ORB) is itself also a very efficient and quick method for comparison.

6.3 Visualization

A visual comparison can give more intuition for the performance of different methods.

Comparing the Figure 4a and Figure 4b, it is obvious that the detected points using method of optical flow are more uniformly distributed. This is because in the implementation, each grid cell of the segmented image is forced to find feature points, which helps to increase the precision of the estimated camera position.

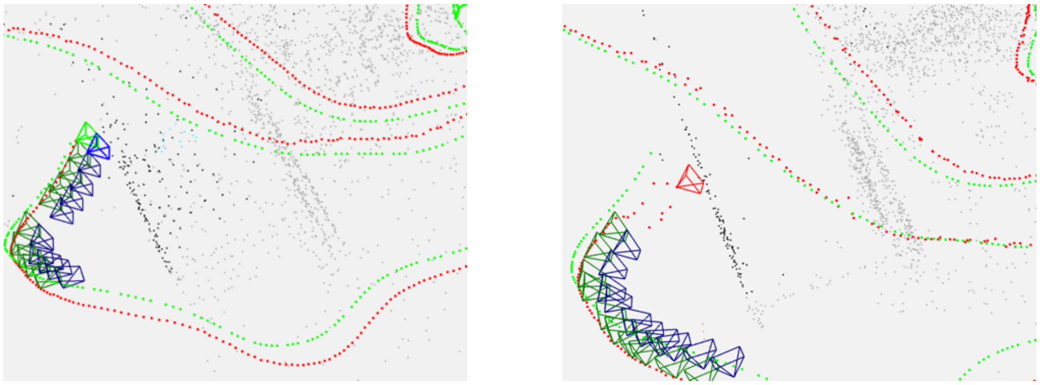
In Figure 5a and Figure 5b, there are two gray lines representing the calibration board in the vertical direction. The two gray lines in Figure 5a are closer than Figure 5b, which means a better reconstructed map in this area. A real world picture of this area is offered in Figure 6.



(a) Optical flow

(b) Descriptors matching

Figure 4: Ground truth path (red) and calculated path (green) using optical flow and descriptor matching methods. Gray points represent detected key points.



(a) Optical flow

(b) Descriptors matching

Figure 5: Map constructed using optical flow or descriptor matching. Gray points represent detected key points.

7 Conclusion

Visual odometry is a very promising research direction, the applications of this technique is very useful in corresponding environment.

This paper reviews some basic concepts in visual odometry, shows the implementation design and details for visual odometry. Specifically, it presents a novel criterion, which is segmenting image into sub-images and counting number of empty cells to choose key frames. The maximum number of empty cells is a parameter that highly influences the performance of the map construction. A too high or too low number will cause decrease in performance, which indicates that the number of key frames should be kept in a suitable number for a good performance. The strategy of key point detection according to sub-images offers more uniformly distributed key points than key point detection according to the whole image. The execution time of different parts is different in different in methods with and without optical flow. For the method with optical flow, the choice of parameter (e.g. max. number of empty cells) also influences the

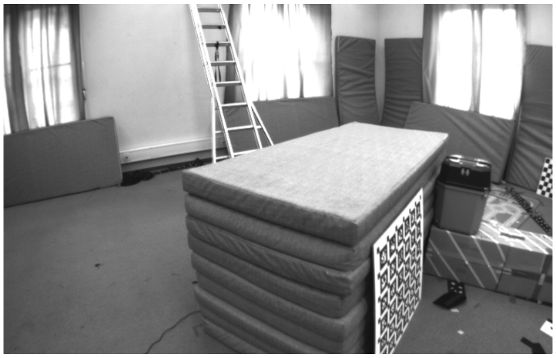


Figure 6: The desk image in real world.

execution time.

References

[1] H. Alismail, ”Visual Odometry for Mobile Robots: Motion Estimation from Imagery,” LAP LAMBERT Academic Publishing, 2010

[2] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha and P. Fua, ”BRIEF: Computing a Local Binary Descriptor Very Fast,” in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 7, pp. 1281-1298, July 2012.

[3] B. D. Lucas and T. Kanade, ”An iterative image registration technique with an application to stereo vision,” In Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI81, pages 674679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[4] B.K.P Horn, B.G. Schunk, ”Determining optical flow, Artificial Intelligence,” vol. 17, pp. 185-203, 1981.

[5] H. Wang and M. Brady, ”Real-time corner detection algorithm for motion estimation, Image and vision computing,” vol. 13, no. 9, pp. 695 703, 1995.

[6] Rodehorst, Volker and Andreas F. Koschan, Comparison and Evaluation of Feature Point Detectors, in Proceedings of 5th International Symposium Turkish-German Joint Geodetic Days, Technical University of Berlin, Germany, March, 2006;

[7] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, ”ORB: an efficientalternative to SIFT or SURF,” in IEEE International Conference on Computer Vision, Barcelona, Spain, November 2011, pp. 25642571

[8] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, ”ORB: An efficient alternative to SIFT or SURF,” 2011 International Conference on Computer Vision, Barcelona, 2011, pp. 2564-2571.

[9] Jianbo Shi and Tomasi, ”Good features to track,” 1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 1994, pp. 593-600.

[10] V. Usenko, N. Demmel, D. Schubert, J. Stueckler and D. Cremers, ”Visual-Inertial Mapping with Non-Linear Factor Recovery,” arXiv:1904.06504, 2019.

Appendices

Our implementation lies all in repository under *s0011*. Our implementation of optical flow visual odometry is in the branch *project_eval*. Visual odometry with descriptor matching (without optical flow) is in the branch *exercise-05_eval*. The report of this project is in the branch *report* in folder ./Report/one_column_version. The presentation slide is in the branch *project_eval* root directory. The optical flow implemented by ourselves is in the branch project_eval_manual.

Algorithm 1 Intensity invariant optical flow

```
initialize pyramid_src[0] with source image;
initialize pyramid_tar[0] with target image;
for all $i \in [1, \mathit{maximum\ level} - 1]$ do
    tmp_src = downsample pyramid_src[i-1];
    tmp_tar = downsample pyramid_tar[i-1];
    pyramid_src.append(tmp_src);
    pyramid_tar.append(tmp_tar);
end for
for all $\mathit{point}$ in $\mathit{source\ points}$ do
    initialize motion = [0,0].transpose();
    initialize dmotion = [0,0].transpose();
    for $l = \mathit{max\ level} - 1$; $i >= 0$; $i - -$ do
        point_scaled = point / pow(2, l);
        img_src = pyramid_src[l];
        img_tar = pyramid_tar[l];
        intensity0 = intensity of source patch scaled by patch mean at img_src;
        gradient0 = [gradient_x, gradient_y].transpose() of source patch at img_src;
        hessian_inverse = sum of hessian.inverse() of source patch at img_src;
        for $i = 0$; $i < \mathit{max\_iteration}$; $i + +$ do
            transformed source patch with motion;
            intensity1 = intensity of transformed patch scaled by patch mean at img_tar;
            for $j = 0$; $j < \mathit{patch\_size}$; $j + +$ do
                residual = intensity1[j] - intensity0[j];
                tmp = gradient0.col[j] * residual;
                dmotion -= hessian_inverse * tmp;
            end for
            motion = motion + dmotion;
            point_scaled_transformed = point_scaled + motion;
            if point_scaled_transformed out of range then
                terminate track for this point;
            end if
            if dmotion.norm() < threshold then
                break iteration;
            end if
        end for
        motion *= 2;
    end for
    output_points.append(point + motion);
end for
```