VISIOTEC Intensity-based Homography Optimization Software: Basic Theory and Use Cases

Lucas Nogueira, Ely C. de Paiva, and Geraldo Silveira

{lucas.nogueira@fem.unicamp.br, elypaiva@fem.unicamp.br, geraldo.silveira@cti.gov.br}

CTI Renato Archer
VISIOTEC research group
Division of Robotics and Computer Vision
Rod. Dom Pedro I, km 143,6, Amarais
CEP 13069-901, Campinas/SP, Brazil

UNICAMP
LEVE laboratory
School of Mechanical Engineering
Rua Mendeleyev, 200, Barão Geraldo
CEP 13083-860, Campinas/SP, Brazil

Technical report CTI-VTEC-TR-01-2017
Revision A

Abstract

This document presents a ready-to-use C++ implementation of a vision-based estimation algorithm. It optimally estimates a projective geometric transformation and global illumination changes between two given images. The geometric part is encoded in a homography matrix, which is key to several applications in computer vision and robotics, such as visual tracking and robot control. Whereas that estimation problem has been traditionally solved using a feature-based scheme, this software solves it using an intensity-based approach. In this case, all possible image information can be exploited, leading to high accuracy levels. At the same time, we are able to achieve a level of computational efficiency that is required by those application domains. For improved versatility, the software provides three different homography classes to choose from. This document encompasses the theory necessary to understand our solution and its limitations, as well as the installation and usage instructions to deploy our software. Furthermore, two of its applications are described: image registration and visual tracking, where the latter is also available as a ROS package. This software has been developed at the VISIOTEC (Vision-based Estimation and Control) group, CTI Renato Archer, for research purposes only. This software is provided "as is" and without any express or implied warranties.

Keywords: Computer vision, robot vision, robotics, image registration, homography estimation.

This work is also partially supported by the Brazilian agency CAPES, and by the National Institute InSAC (CNPq under grant number 465755/2014-3 and FAPESP under grant number 2014/50851-0).

Contents

1 Introduction

A key concept in computer vision is that of a planar homography. It relates corresponding pixel coordinates of a plane in different images. The homography induced by a plane encodes its structure and the camera motion, and has been used in a variety of vision-based applications. Examples of such applications are visual servoing [1,2], visual tracking [3], and image mosaicking [4], which all perform a homography estimation step. Therefore, estimating a homography from a pair of images is a fundamental task in that field.

The homography estimation task can be formulated as an image registration problem. This problem is defined as a search for the parameters that best define a transformation between corresponding pixels in a pair of images. Solutions to this problem involve the definition of four important characteristics: the information space; the transformations models; the similarity measures; and the search strategy [5].

In the information space we find two main approaches: feature- and intensity-based. The former requires the extraction and association of geometric primitives (e.g., point, lines, ellipses, etc.) in different images before the actual estimation can occur. The latter simultaneously solves for the estimation problem and for the pixel correspondences. The algorithm presented in this document uses the intensity-based approach.

The transformation model used in this software is composed of a geometric and a photometric part [3]. The homography matrix is the parameter that models the geometric transformation. The photometric parameters are derived from illumination models that can vary greatly in its complexity. In this work, we estimate only global illumination changes. By integrating the photometric parameters into the estimation, we also increase the accuracy of the geometric parameters estimation.

The search strategy adopted here is a multidimensional optimization problem. The goal is to find an optimal set of parameters that minimizes a given similarity measure. In our case, this measure is the sum of squared differences over the pixel intensities. To solve this problem, an initial solution is iteratively refined using a nonlinear optimization method. This software applies the efficient second-order method, which increases the rate as well as the domain of convergence of the optimization. Nevertheless, the initial solution still needs to be carefully chosen given its local operation. This issue is generally overcome with a predictor (this software provides a correlation-based one) or by requiring small interframe displacements [6].

This document presents a ready-to-use C++ implementation of an intensity-based homography estimation algorithm. Three classes of homography with different degrees of freedom are provided. This improves versatility to handle specific situations properly. In Section 2, the basic theoretical background necessary for understanding our solution and its limitations is presented. Sections 3 and 4 describe the software architecture and its installation procedures, respectively. Finally, Section 5 contains two common practical applications: image registration and visual tracking, where the latter is also available as a ROS package.

2 Theoretical Background

This section introduces the basic theory concepts to understand our solution. It directly determines the working conditions of our algorithm, which are also summarized here.

2.1 Image registration

The goal of the image registration problem is to estimate the transformation that optimally aligns two images of the same scene. Pixels in the different images that correspond to the same scene point are then mapped to each other. In this document, we refer to the first image as the reference image, and the second one as the current image. Existing methods to solve this problem can be viewed as a combination of four components: (1) information space; (2) transformation model; (3) similarity measure; and (4) search strategy. In the next sections, we present each one of these components, and how our algorithm approaches them.

2.1.1 Information space

The information space dictates what information from the images is exploited in the registration. In this sense, most existing methods can be classified into two categories:

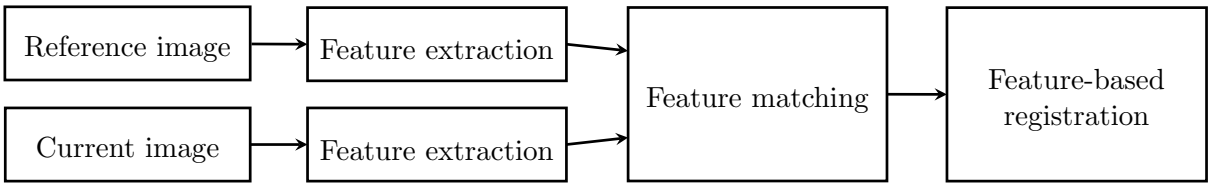
Feature-based In this category, geometric primitives (e.g., points, lines, ellipses, etc.), also known as image features, are extracted from each image. The information space regards the parametrization of these features in the images. Primitives from different images are compared and matched to obtain correspondences between them. The set of corresponding features is the input to the image registration algorithm. These methods thus depend heavily on the accuracy of the extraction and matching steps. They can even fail due to the outliers in these steps.

Intensity-based In this category, the image registration algorithm directly exploits the pixel intensities, with no intermediate steps. The information space regards the intensity values of the pixels in the image. They are also known as direct methods, appearance-based, and texture-based. Intensity-based methods can achieve high levels of accuracy due to the fact that they can exploit all image information. These methods often assume a sufficient overlapping between the two images.

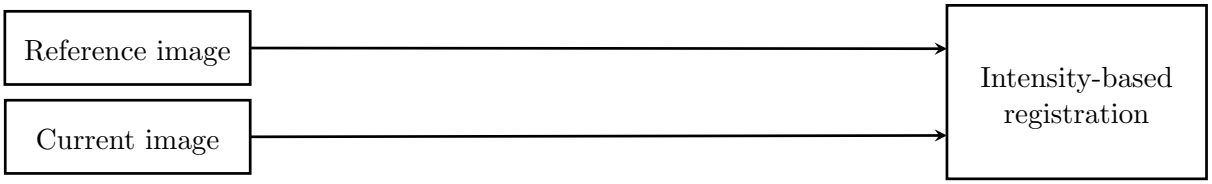
The image registration pipeline for both approaches are shown in Fig. 1. Our software applies the latter approach.

2.1.2 Image transformations models

These models define the admissible transformations that can be applied to the images. The basic idea is to have models that are flexible enough to account for most of the changes that



(a) Feature-based



(b) Intensity-based

Figure 1: Feature- vs intensity-based registration pipeline.

occur in the image, while also balancing the complexity involved. Therefore, it is fundamental to properly define the transformation models. This software applies geometric and photometric transformations, which are presented below.

Geometric transformation Geometric transformations model image changes due to variations in the scene structure and/or the camera motion. For a given pixel in the reference frame, we model its change of position due to geometric factors as

$$\mathbf{p} \propto \mathbf{H}\,\mathbf{p}^*, \tag{1}$$

where the symbol "$\propto$" denotes proportionality up to a nonzero scale factor, $\mathbf{p}^*$ is the homogeneous pixel coordinates in the reference image, $\mathbf{p}$ is its corresponding coordinates in the current image, and the (3×3)-matrix $\mathbf{H}$ is the projective homography. Equation (1) is appropriate to model geometric transformations when at least one out of the three conditions below holds:

- Planar surfaces: The observed object is planar, with no constraints on the camera motion;
- Objects at infinity: The scene is far from the camera, with no constraints on the scene geometry or on the camera motion;
- Purely rotational motion: The camera undergoes a pure rotation between images, with no constraints on the scene geometry.

The homography matrix $\mathbf{H}$ is related to the scene structure and camera motion via

$$\mathbf{H} \propto \mathbf{K}\left(\mathbf{R} + \frac{1}{d^*}\mathbf{t}\mathbf{n}^{*\top}\right)\mathbf{K}^{-\mathbf{1}}, \tag{2}$$

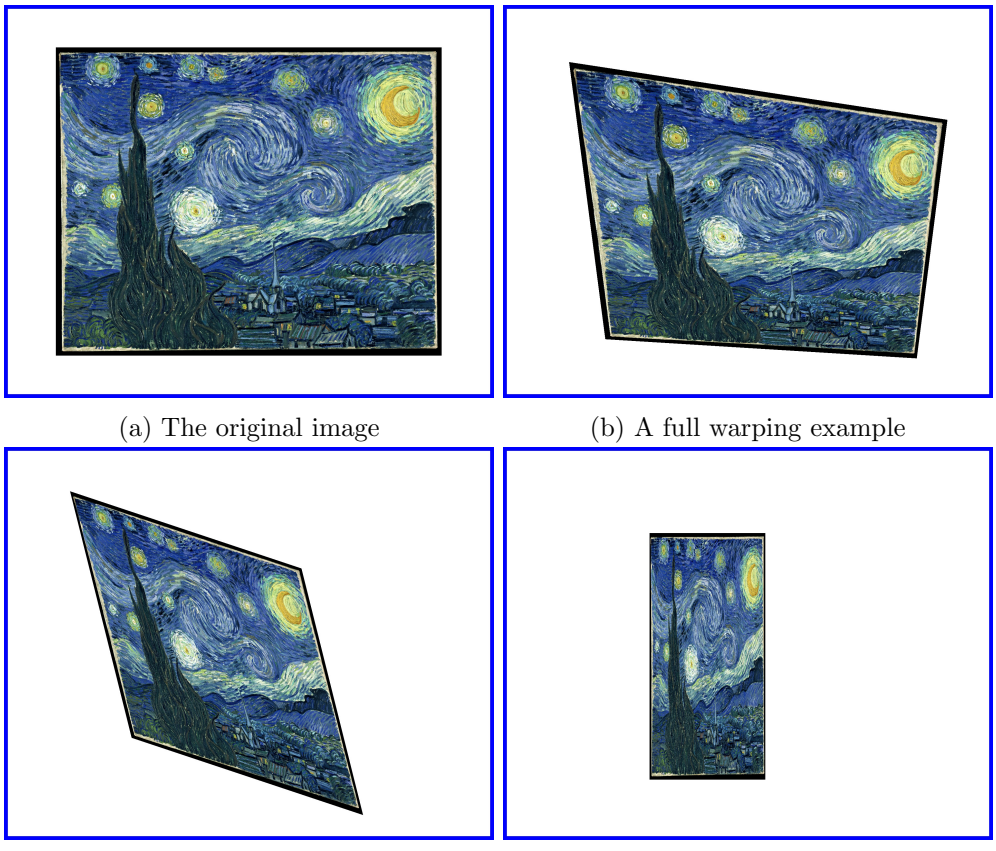
where $\mathbf{K} \in \mathbb{R}^{3\times3}$ contains the camera intrinsic parameters; $\mathbf{R} \in \mathbb{SO}(3)$ and $\mathbf{t} \in \mathbb{R}^3$ are the rotation matrix and translation vector associated with the camera motion, respectively; and $d^* > 0$ and

$\mathbf{n}^* \in \mathbb{R}^3 : \|\mathbf{n}^*\| = 1$ are, respectively, the distance to the surface plane and its normal vector with respect to the reference frame. The decomposition of $\mathbf{H}$ into those Euclidean components is out of the scope of this software.

This software provides three homography categories. Each category is related to the number of degrees of freedom (d.o.f.) of the image transformation:

1. Full: This is the most general homographic transformation. Thus, it contains eight d.o.f.;
2. Affine: This homography contains six d.o.f., which are related to the two image translations (vertical and horizontal), the two scalings, a rotation and a shear transformation;
3. Stretch: This homography contains four d.o.f., which are related to the two image translations and the two scaling transformations.

Examples of each homography category are shown in Fig. 2. Depending on the application at hand, the user can choose a particular homography to constrain the transformation model and thus improve the results.



(a) The original image (b) A full warping example

(c) An affine warping example (d) A stretch warping example

Figure 2: Examples of different homographic transformations applied to Van Gogh's *Starry Night*.

Photometric transformation This transformation model aims to explain the changes in the image due to variations in the lighting conditions of the scene. This software models only global illumination changes, i.e., changes that apply equally to all pixel in the images. This model can be defined as

$$\mathcal{I}'(\mathbf{p}) = \alpha\,\mathcal{I}(\mathbf{p}) + \beta, \tag{3}$$

where $\mathcal{I}(\mathbf{p})$ is the intensity value of the pixel $\mathbf{p}$, $\mathcal{I}'(\mathbf{p})$ denotes its transformed intensity, and the gain α and the bias β are the parameters that fully define the transformation. These parameters represent the adjustments in the image constrast and brightness, respectively.

2.1.3 Similarity measures

Similarity measures give an indication of the registration quality. This software applies two of these measures, as described below.

SSD The first similarity measure is the Sum of Squared Differences over the pixel intensities:

$$\mathrm{SSD}(\mathcal{I}', \mathcal{I}^*) = \sum_i \left[\, \mathcal{I}'(\mathbf{p}_i, \mathbf{x}) - \mathcal{I}^*(\mathbf{p}_i^*) \,\right]^2, \tag{4}$$

where $\mathcal{I}^*(\mathbf{p}_i^*)$ is the intensity value of $\mathbf{p}_i^*$ in the reference image, $\mathcal{I}'(\mathbf{p}_i, \mathbf{x})$ is the intensity value of the current image photogeometrically transformed using the parameters $\mathbf{x} = \{\mathbf{H}, \alpha, \beta\}$. The SSD is used in the registration as the cost function to be minimized. Because the optimal transformation results in a minimal value, the SSD is technically a dissimilarity measure.

ZNCC The other similarity measure used to assess the registration quality is the Zero Normalized Cross Correlation:

$$\mathrm{ZNCC}(\mathcal{I}^*, \mathcal{I}') = \left\langle \frac{\mathcal{I}_v^* - \bar{\mathcal{I}}_v^*}{\left\|\mathcal{I}_v^* - \bar{\mathcal{I}}_v^*\right\|}, \frac{\mathcal{I}_v' - \bar{\mathcal{I}}_v'}{\left\|\mathcal{I}_v' - \bar{\mathcal{I}}_v'\right\|} \right\rangle, \tag{5}$$

where $(\cdot)_v$ denotes the vectorized form of an image, $\bar{(\cdot)}$ represents its mean value, and $\langle \cdot, \cdot \rangle$ is the inner product. This similarity measure varies between -1 (very bad) and $+1$ (perfect). Especially because of such normalizations, the ZNCC is useful both for evaluating the final quality of the estimation and as a sliding window predictor.

2.1.4 Search strategy

This registration software formulates the search strategy as a nonlinear optimization problem.

Optimization solution To solve the underlying optimization problem from (4), we apply the Efficient Second-order Minimization method. Its advantages when registering images include both a higher convergence rate and a larger convergence domain than classical first-order methods, with the absence of costly Hessian calculations [6].

Solution improvements We also make use of a multiresolution pyramid. This strategy consists of creating a sequence of smaller templates until a predefined minimum size is reached. Two pyramids are built, one for the current template and another for the reference template. This software solves the optimization from the lowest resolution templates to the highest ones. The benefits of this coarse-to-fine strategy include an increase in the computational efficiency, avoidance of spurious local minima, and a larger domain of convergence.

2.2 Working conditions

The presented theory leads to a set of working conditions for the software. In the case of a "Full" homography estimation (see Fig. 2), these conditions are summarized below:

- For unknown camera motion, the observed object is planar or is at infinity;
- For pure rotational camera motion, the observed object can be of any shape and distance;
- The observed object is sufficiently textured;
- The observed object is subject only to global illumination changes;
- The observed object is occlusion-free;
- The inter-frame displacement is relatively small.

3 Software Architecture

Our software provides a C++ library that implements the homography optimization using the theoretical principles previously described. The library core features are contained in the **IBGHomographyOptimizer** class. This is an abstract base class that defines the optimizer main behavior. The IBG is an acronym for *I*ntensity-*B*ased technique robust to *G*lobal illumination changes.

We offer three variants of the algorithm for improved versatility. Indeed, each variant relates to specific projective motion constraints as described in Section 2.1.2. These variants are implemented as derived classes using the keywords **Full**, **Affine** or **Stretch**. This architecture allows the application developer to use the same interface for all those categories, only changing the object declaration. Figure 3 shows the Unified Modeling Language (UML) diagram for our software.

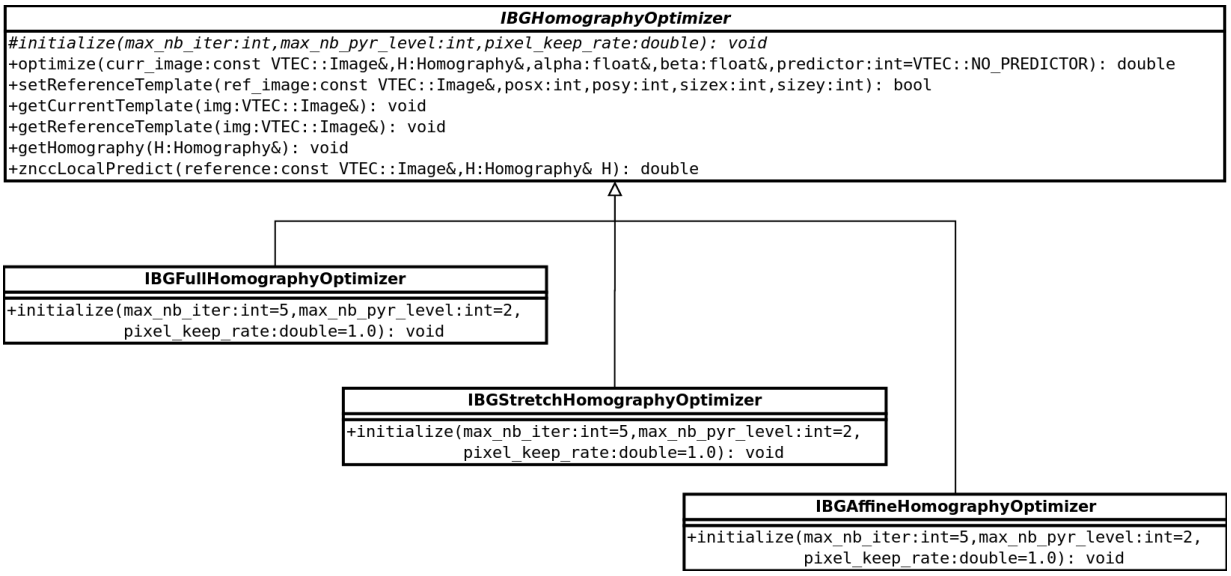


Figure 3: UML model of our homography optimization software.

4 Software Installation

Our software is hosted on the Github version control repository and Internet hosting platform. It can be found and downloaded using the link:

`https://github.com/lukscasanova/vtec`

We have also made available a ROS package [7] that implements a visual tracker using our presented libraries. This package can be found in:

`https://github.com/lukscasanova/vtec_ros`

which contains its documentation and videos as well. The remainder of this report is related only to the core C++ libraries.

4.1 Minimum requirements

Before you install our software, make sure that your platform meets the following minimum software requirements:

- GCC version 5.4.1 or later
- CMake version 2.8.3 or later
- Git

4.2 Compiling instructions

To compile the latest codebase (if a release version has been downloaded, you may skip the first command below), open a terminal and issue the following commands:

```
# Clone the repository
$ git clone https://github.com/lukscasanova/vtec.git

# Enter the directory
$ cd vtec

# Create and enter the build directory
$ mkdir build
$ cd build

# Configure cmake
$ cmake ..

# Compile
$ make

# Return to base directory
$ cd ..
```

5 Use Cases

This section presents two applications of our software. We show both how to execute them and how the code is organized in each situation.

5.1 Homography-based image registration

In this application, the goal is to find the homography that best relates patches of two given images. From the base directory, we execute the following command:

```
$./build/vtec_ibg_optimize_homography_example
```

This use case takes 2 images from our sample dataset (located in the **seq** folder), and runs the homography estimation method. We know beforehand where in the first image the planar surface of interest is located, pixelwise. The estimation will then provide us with the location of that planar surface in the second image, the homography matrix, and photometric parameters. The code of this use case is listed on Appendix A. We present below its critical steps.

First, we create the homography optimizer object, and call the **initialize** method.

```
/* Tracker initialization without reference image */
VTEC::IBGFullHomographyOptimizer ibg_optimizer;
ibg_optimizer.initialize(MAX_NB_ITERATION_PER_LEVEL,
    MAX_NB_PYR_LEVEL, PIXEL_KEEP_RATE);
```

Notice that we pass three parameters to the **initialize** method. They configure the optimization method, and are described below:

MAX_NB_ITERATION_PER_LEVEL It defines the maximum number of optimization iterations in each level of the multi-resolution pyramid. Default value is 5. The higher this number, the higher the computational effort.

MAX_NB_PYR_LEVEL It defines the maximum number of pyramid levels to be used. Default value is 2. A value of 1 will make the optimizer use only the lowest possible resolution level. Therefore, the software will be faster but less precise. In the multi-resolution pyramid, the software stops creating new pyramid levels if the downsized image size is not large enough. Increasing this value will increase the resolution used, up to the actual resolution of the region of interest.

SAMPLING_RATE It defines how the image is sampled. This value ranges from 0 to 1. Default values is 1.0, which means all pixels are used. The method always uses a regular sampling of the image pixels. A good lower bound for this parameter is 0.5, which means we keep 50% of the pixels.

Next, we load the first image, i.e., the reference image. This image is used to extract the reference template, which corresponds to the region of interest. To do this, we use the **setReferenceTemplate** method.

```
/* Load reference image */
VTEC::Image reference_image;
std::ostringstream refFileNameStream;
refFileNameStream << directory << file_prefix <<
    std::setw(digit_width) << std::setfill('0') << 0 << ".pgm";
reference_image.loadPGM(refFileNameStream.str());
ibg_optimizer.setReferenceTemplate(reference_image, BBOX_POS_X,
    BBOX_POS_Y, BBOX_SIZE_X, BBOX_SIZE_Y);
```

The method uses the following parameters to extract the template:

BBOX_POS_X, BBOX_POS_Y It defines the position of the upper left corner of the bounding box in pixel coordinates.

BBOX_SIZE_X, BBOX_SIZE_Y It defines the length of the sides of the bounding box in pixels.

Then, we load the second (current) image.

```
/* Load current image */
VTEC::Image current_image;
std::ostringstream currFileNameStream;
currFileNameStream.clear();
currFileNameStream << directory << file_prefix <<
    std::setw(digit_width) << std::setfill('0') << 2 << ".pgm";
current_image.loadPGM(currFileNameStream.str());
```

As an input to the optimization function, we need to set the parameters that will be used as a first guess in the optimization method. Because we have no prior knowledge other than the position of the planar surface in the first image, we can use that as our first guess.

```
/* Optimization parameters */
VTEC::Homography H;
float alpha = 1.0;
float beta  = 0.0;

/* Initialize Homography first guess */
H[0] = 1.0;H[1] = 0.0; H[2] = BBOX_POS_X;
H[3] = 0.0;H[4] = 1.0; H[5] = BBOX_POS_Y;
H[6] = 0.0;H[7] = 0.0; H[8] = 1.0;
```

Finally, we call the optimization function. Because we have already set the reference template, we only need to pass the current image to the optimization method. The parameters H, *alpha* and *beta* will be updated with the estimated values once the optimization is finished.

```
/* optimize function */
score = ibg_optimizer.optimize(current_image, H, alpha, beta,
    VTEC::NO_PREDICTOR);
```

The last parameter of the optimization function sets the type of predictor to be used. In this use case, we do not use any predictor (refer to the next use case to understand more about the predictors). Also, the function returns the ZNCC score associated with the optimization result. This value gives the user an idea of the quality of the estimation. Ideally, this value should be very close to 1.

We can use the homography optimizer auxiliary functions to obtain the templates that were built during the optimization. The reference template is the region of interest obtained by directly applying the bounding box to the reference image. The current template is the result of warping the current image using the estimated geometric parameters. Those two images should be nearly equal.

```
VTEC::Image current_template;
ibg_optimizer.getCurrentTemplate(current_template);

VTEC::Image reference_template;
ibg_optimizer.getReferenceTemplate(reference_template);
```

The terminal will print out the resulting ZNCC score. In the **res** folder, three images will be generated:

reference_pattern.pgm This is the pattern as selected from the reference image using the bounding box position and size;

annotated_reference_image.pgm This is the first image annotaded with the original location of the reference pattern;

annotated_current_image.pgm This is the second image annotaded with the new location of the reference pattern;

warped_pattern.pgm This is the current pattern warped using the estimated homography. Ideally, it matches closely the reference_pattern.

5.2 Homography-based visual tracking

In this use case, the goal is to build an image tracking application. Starting from the first image (i.e., the reference one), we want to track a region of interest throughout an image stream. The estimation from the previous image becomes the initialization estimate for the current image. It is also possible to use a predictor (which is described later) to attempt to improve this estimate. From the base directory, we execute the following command:

```
$./build/vtec_ibg_tracker_example
```

In this tracker application (its code is listed on Appendix B), our homography optimizer will use the same reference image when solving the optimization problem for each image in the sequence. Hence we only need to call **setReferenceImage** once for the whole sequence.

```
/* Tracker initialization with reference image */
/* Choose between the three variants Full, Strectch and Affine */
VTEC::IBGFullHomographyOptimizer myTracker;
myTracker.initialize(MAX_NB_ITERATION_PER_LEVEL, MAX_NB_PYR_LEVEL,
    PIXEL_KEEP_RATE);
myTracker.setReferenceTemplate(I, BBOX_POS_X, BBOX_POS_Y,
    BBOX_SIZE_X, BBOX_SIZE_Y);
```

Hence, we are able to obtain the reference template immediately.

```
/* Save the tracked pattern */
VTEC::Image reference_template;
myTracker.getReferenceTemplate(reference_template);
reference_template.savePGM(directory + "/res/patr.pgm");
```

We call the optimization function for each new image in the sequence, and feed the results from the last image to the current image. However, for the first image after the reference image,

we use the parameters that are directly derived from the reference image and the bounding box we set. This is similar to the approach used in the previous use case, but it is different in that we can now obtain the homography directly from the optimizer object.

```
/* Optimization parameters */
VTEC::Homography H;
float alpha = 1.0;
float beta  = 0.0;
myTracker.getHomography(H);
```

These next steps are repeated for each new accessed image in the sequence (the code in Appendix B jumps every other image only to showcase the predictor's function). First, we load the current image.

```
    /* load next image in the sequence */
    std::ostringstream fileNameStream;
    fileNameStream << directory << file_prefix <<
        std::setw(digit_width) << std::setfill('0') <<
        current_image_number << ".pgm";
    std::cout << "processing image: " << fileNameStream.str() <<
        std::endl;

    if(I.loadPGM(fileNameStream.str()) == false){
        std::cout << "Unable to load next file. Exiting..." <<
            std::endl;
        break;
    };
```

Next, we call the optimize function, which returns the ZNCC score associated with the estimated parameters. The parameters (H, *alpha* and *beta*) will be updated with the estimated values once the optimization is finished.

```
    score = myTracker.optimize(I, H, alpha, beta,
        VTEC::ZNCC_PREDICTOR);
```

The chosen predictor uses a sliding window approach to scan the image using the ZNCC score in an attempt to improve the homography estimate before actually executing the optimization. This helps increasing the convergence domain of the tracking application.

As in the previous use case, the terminal will print out the ZNCC score associated with the optimal estimate. This can act as a confidence metric to help the user in identifying good and bad results. In the **res** folder you will find:

patr.pgm It is the original pattern as selected from the first image using the chosen parameters.

im*.pgm It represents the sequence of original images with the warped bounding box of the Region Of Interested (ROI) annotated in each one. This will show the user how the ROI

moved between frames.

pc*.pgm It represents the sequence of warped patterns obtained from applying the estimated homography. Ideally, it should change very little (aside from illumination changes), and match the original pattern;

Let us remind that this tracking application has been further integrated into a ROS package, which can be found in:

`https://github.com/lukscasanova/vtec_ros`

which also contains its documentation and videos.

Appendices

A Listing of vtec_ibg_optimize_homography_example.cpp

```
#include <vtec_core/image.h>
#include <vtec_core/draw.h>
#include <homography_optimizer/ibg_interface.h>
#include <iostream>
#include <sstream>
#include <iomanip>
#include <string>

/* Bounding box left-top position  (x,y) in the first image */
#define BBOX_POS_X                    200
#define BBOX_POS_Y                    250

/* Bounding box size (x, y) in the first image  */
#define BBOX_SIZE_X                   200
#define BBOX_SIZE_Y                   200

/* Maximum number of iterations per pyramid level. Lower means less computation
 ↪  effort */
#define MAX_NB_ITERATION_PER_LEVEL    5

/* Number of levels in the pyramid, use more if you need more precision */
#define MAX_NB_PYR_LEVEL              2

/* Sampling rate, 1.0 means 100% of points are used */
#define PIXEL_KEEP_RATE               1.0

int main(int argc, char** argv){
   std::string directory = ".";
   std::string file_prefix = "/seq/im";
   int digit_width = 3;

   /* Tracker initialization without reference image */
   VTEC::IBGFullHomographyOptimizer ibg_optimizer;
   ibg_optimizer.initialize(MAX_NB_ITERATION_PER_LEVEL, MAX_NB_PYR_LEVEL,
    ↪  PIXEL_KEEP_RATE);

   /* Load reference image */
   VTEC::Image reference_image;
   std::ostringstream refFileNameStream;
```

```
38    refFileNameStream << directory << file_prefix <<  std::setw(digit_width) <<
      ↪  std::setfill('0') << 0 << ".pgm";
39    reference_image.loadPGM(refFileNameStream.str());
40    ibg_optimizer.setReferenceTemplate(reference_image, BBOX_POS_X, BBOX_POS_Y,
      ↪  BBOX_SIZE_X, BBOX_SIZE_Y);
41
42    /* Load current image */
43    VTEC::Image current_image;
44    std::ostringstream currFileNameStream;
45    currFileNameStream.clear();
46    currFileNameStream << directory << file_prefix <<  std::setw(digit_width) <<
      ↪  std::setfill('0') << 2 << ".pgm";
47    current_image.loadPGM(currFileNameStream.str());
48
49    /* Optimization parameters */
50    VTEC::Homography H;
51    float alpha = 1.0;
52    float beta  = 0.0;
53
54    /* Initialize Homography first guess */
55    H[0] = 1.0;H[1] = 0.0; H[2] = BBOX_POS_X;
56    H[3] = 0.0;H[4] = 1.0; H[5] = BBOX_POS_Y;
57    H[6] = 0.0;H[7] = 0.0; H[8] = 1.0;
58
59    /* Draws the region of interest in the reference image */
60    drawResult(BBOX_SIZE_X, BBOX_SIZE_Y, H, reference_image);
61
62    /* optimization stats */
63    double score;
64
65    /* optimize function */
66    score = ibg_optimizer.optimize(current_image, H, alpha, beta,
      ↪  VTEC::NO_PREDICTOR);
67
68    if(score == -1.0){
69       std::cout << "optimization failed" << std::endl;
70       return 0;
71    }else{
72       std::cout << "Score: " << score << std::endl;
73    }
74
75    /* Save the results */
76    drawResult(BBOX_SIZE_X, BBOX_SIZE_Y, H, current_image);
77
78    VTEC::Image current_template;
```

```
79      ibg_optimizer.getCurrentTemplate(current_template);
80
81      VTEC::Image reference_template;
82      ibg_optimizer.getReferenceTemplate(reference_template);
83
84      /* Save images, annotated and the current template warped */
85      std::string annotated_current_image_path, annotated_reference_image_path,
        ↪  current_template_path, reference_template_path;
86      annotated_current_image_path = directory +
        ↪  "/res/annotated_current_image.pgm";
87      annotated_reference_image_path = directory +
        ↪  "/res/annotated_reference_image.pgm";
88      current_template_path = directory + "/res/warped_template.pgm";
89      reference_template_path = directory + "/res/reference_template.pgm";
90
91      current_image.savePGM(annotated_current_image_path);
92      current_template.savePGM(current_template_path);
93      reference_template.savePGM(reference_template_path);
94      reference_image.savePGM(annotated_reference_image_path);
95
96      return 0;
97  }
```

B Listing of vtec_ibg_tracker_example.cpp

```
1   #include <homography_optimizer/ibg_interface.h>
2   #include <vtec_core/image.h>
3   #include <vtec_core/draw.h>
4   #include <iostream>
5   #include <sstream>
6   #include <iomanip>
7   #include <string>
8
9   /* Bounding box left-top position  (x,y) in the first image */
10  #define BBOX_POS_X                  250
11  #define BBOX_POS_Y                  200
12
13  /* Bounding box size (x, y) in the first image  */
14  #define BBOX_SIZE_X                 200
15  #define BBOX_SIZE_Y                 200
16
17  /* Maximum number of iterations per pyramid level. Lower means less computation
    ↪  effort */
18  #define MAX_NB_ITERATION_PER_LEVEL    5
```

```
19
20 /* Number of levels in the pyramid, use more if you need more precision */
21 #define MAX_NB_PYR_LEVEL              2
22
23 /* Sampling rate, 1.0 means 100% of points are used */
24 #define PIXEL_KEEP_RATE               1.0
25
26 int main(int argc, char** argv){
27
28    std::string directory = ".";
29    std::string file_prefix = "/seq/im";
30    int digit_width = 3;
31
32    /* Use first image in the sequence to initialize tracker*/
33    VTEC::Image I;
34
35    std::ostringstream fileNameStream;
36    fileNameStream << directory << file_prefix <<  std::setw(digit_width) <<
      ↪  std::setfill('0') << 0 << ".pgm";
37
38    I.loadPGM(fileNameStream.str());
39
40    /* Tracker initialization with reference image */
41    /* Choose between the three variants Full, Strectch and Affine */
42    VTEC::IBGFullHomographyOptimizer myTracker;
43    myTracker.initialize(MAX_NB_ITERATION_PER_LEVEL, MAX_NB_PYR_LEVEL,
      ↪  PIXEL_KEEP_RATE);
44    myTracker.setReferenceTemplate(I, BBOX_POS_X, BBOX_POS_Y, BBOX_SIZE_X,
      ↪  BBOX_SIZE_Y);
45
46    /* Save the tracked pattern */
47    VTEC::Image reference_template;
48    myTracker.getReferenceTemplate(reference_template);
49    reference_template.savePGM(directory + "/res/patr.pgm");
50
51    /* Run the tracker and print result */
52    int current_image_number = 0;
53    VTEC::Image current_template;
54
55    /* Optimization parameters */
56    VTEC::Homography H;
57    float alpha = 1.0;
58    float beta  = 0.0;
59    myTracker.getHomography(H);
60
```

```
61    /* optimization stats */
62    double score;
63
64    /* loop through the images in a sequence */
65    while(current_image_number < 10){
66
67       /* load next image in the sequence */
68       std::ostringstream fileNameStream;
69       fileNameStream << directory << file_prefix <<  std::setw(digit_width) <<
          ↪  std::setfill('0') << current_image_number << ".pgm";
70       std::cout << "processing image: " << fileNameStream.str() << std::endl;
71
72       if(I.loadPGM(fileNameStream.str()) == false){
73          std::cout << "Unable to load next file. Exiting..." << std::endl;
74          break;
75       };
76
77       /* run the optimization and record time*/
78       score = myTracker.optimize(I, H, alpha, beta, VTEC::ZNCC_PREDICTOR);
79
80       if(score == -1.0){
81          std::cout << "optimization failed" << std::endl;
82          break;
83       }else{
84          std::cout << "Score: " << score << std::endl;
85       }
86
87       myTracker.getCurrentTemplate(current_template);
88
89       /* Draw a bounding box using the calculated homography onto the image */
90       drawResult(BBOX_SIZE_X, BBOX_SIZE_Y, H, I);
91
92       /* Save annotated images and the current pattern warped */
93       std::string annotaded_image_path, current_template_path;
94       char filename_string[200];
95       sprintf(filename_string, "%s/res/im%d.pgm", directory.c_str(),
          ↪  current_image_number);
96       annotaded_image_path = std::string(filename_string);
97
98       sprintf(filename_string, "%s/res/pc%d.pgm", directory.c_str(),
          ↪  current_image_number);
99       current_template_path = std::string(filename_string);
100
101      I.savePGM(annotaded_image_path);
102      current_template.savePGM(current_template_path);
```

```
103
104         /* Jump ahead in the sequence */
105         current_image_number += 2;
106     }
107
108     return 0;
109 }
```

References

[1] G. Silveira, L. Mirisola, and P. Morin. Decoupled direct visual servoing. In *IEEE/RSJ IROS*, pages 71–76, Japan, 2013.

[2] G. Silveira. On intensity-based nonmetric visual servoing. *IEEE Transactions on Robotics*, 30(4):1019–1026, 2014.

[3] G. Silveira and E. Malis. Unified direct visual tracking of rigid and deformable surfaces under generic illumination changes in grayscale and color images. *International Journal of Computer Vision*, 89:84–105, 2010.

[4] O. Faugeras, Quang-Tuan Luong, and Theo Papadopoulo. *The geometry of multiple images*. The MIT Press, 2001.

[5] L. Brown. A survey of image registration techniques. *ACM computing surveys*, 24(4):325–376, 1992.

[6] S. Benhimane and E. Malis. Homography-based 2D visual tracking and servoing. *The International Journal of Robotics Research*, 26(7):661–676, 2007.

[7] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: An open-source robot operating system. In *ICRA workshop on open source software*, 2009.