VisioTec robust intensity-based homography optimization software

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-01-19
Revision A

Abstract

This document presents a ready-to-use C++ implementation of a vision-based estimation algorithm. It optimally and robustly estimates a projective geometric transformation and global illumination changes between two 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, as well as of robustness to global lighting variations and to unknown occlusions. For improved versatility, the software provides three different homography classes to choose from. This document describes 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, homography estimation, robust methods, image registration, visual tracking.

This work was supported in part by the Coordination for the Improvement of Higher Education Personnel (CAPES) under Grant 88887.136349/2017-00, in part by the Sao Paulo Research Foundation (FAPESP) under Grant 2017/22603-0, and in part by the National Institute InSAC (CNPq under Grant 465755/2014-3 and FAPESP under Grant 2014/50851-0). G. Silveira would also like to thank Prof. Markus Vincze at TUW, Vienna, Austria, for his support during a sabbatical stay at his group.

Contents

1 Introduction

The homography matrix is a key component in computer vision. 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 at least four important characteristics [5]: The information space; the transformations models; the similarity measures; and the search strategy.

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. 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 relatively small interframe displacements [6].

This document presents a ready-to-use C++ implementation of a robust intensity-based homography estimation algorithm. Three classes of homography with different degrees of freedom are provided. This improves versatility to handle specific situations properly. As an optional mode, the robustness to unknown occlusions can also be actived. 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 and, optionally, of a robust method:

1. information space;
2. transformation model;
3. similarity measure;
4. search strategy; and
5. robust method.

Next sections present each one of these components, and how our algorithm approaches them.

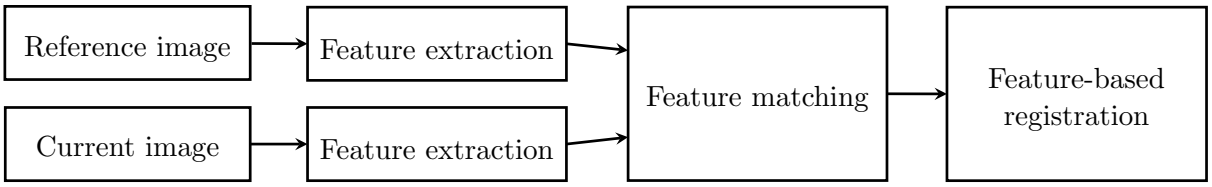
2.1.1 Information space

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:

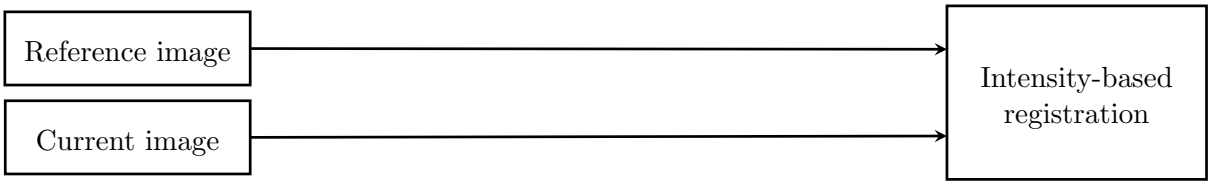
Feature-based In this category, geometric primitives (e.g., points, lines, ellipses, etc.), also known as image features, are extracted from each image. The information space regards the parametrization of these features in the images. Primitives from different images are compared and matched to obtain 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.



(a) Feature-based



(b) Intensity-based

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

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 occur in the image, while also balancing the complexity involved. Therefore, it is fundamental to properly define the transformation models. Our software combines geometric and photometric transformations. By considering the illumination changes jointly with the homography, the estimation is made inherently robust to these changes.

Geometric transformation The geometric transformations model explains 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 in the current one as

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

where $\mathbf{H} \in \mathbb{SL}(3)$ is the projective homography matrix, $\mathbf{p}^* \in \mathbb{P}^2$ is the homogeneous pixel coordinates in the reference image, and $\mathbf{p} \in \mathbb{P}^2$ is its corresponding coordinates in the current image. 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

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

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 to improve the results.

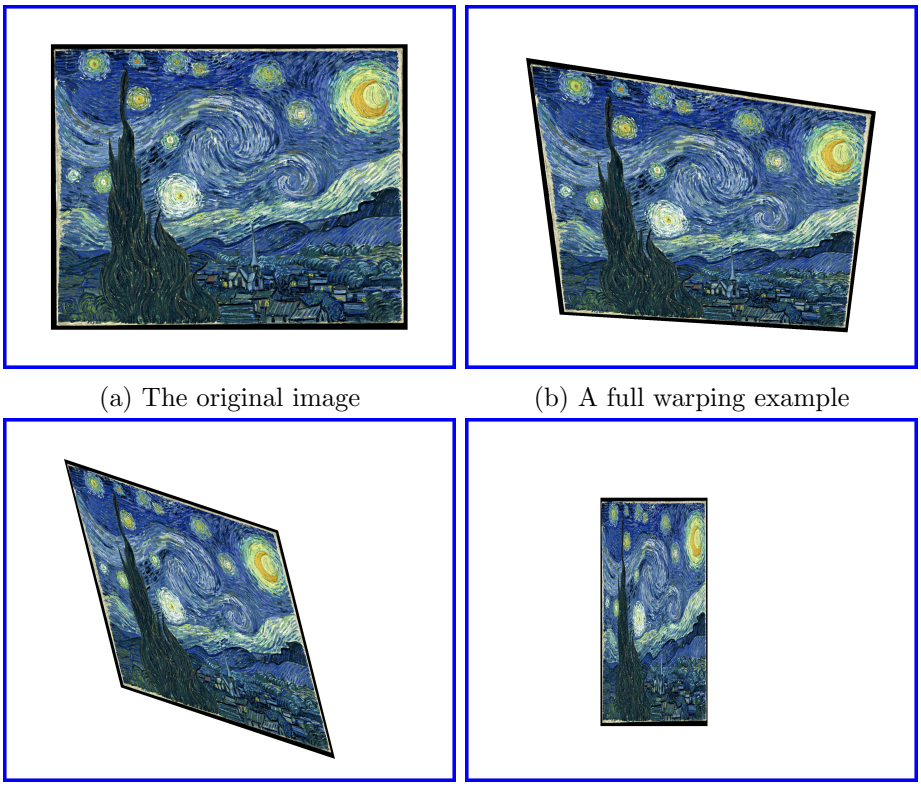
Photometric transformation This transformation model aims to explain the changes in the image due to variations in the lighting conditions of the scene. This software models only global illumination changes, i.e., changes that apply equally to all pixel in the images. This model can be defined as

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

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 can be viewed as 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.



(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*.

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 commonly used in image 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 software formulates the search strategy as a nonlinear optimization problem.

Optimization solution To solve the nonlinear Least Squares optimization problem using (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 standard iterative methods, with no Hessian calculations. Let us note that this software thus implements the Case 3.1.1 along with Case 3.2.1 of [3].

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 one 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.

2.1.5 Robust method

Real-world applications often require the use of a robust estimation method to deal with model uncertainties and measurement errors (outliers), such as unknown occlusions. Least Squares algorithms are not robust since in theory a single outlier can lead to an estimate arbitrarily far from the true solution.

M-estimators The robust equivalent of the Least Squares family are the M-estimators. In this case the cost function to be minimized is modified to

$$\sum_i \rho\big(r_i(\mathbf{x})\big), \tag{6}$$

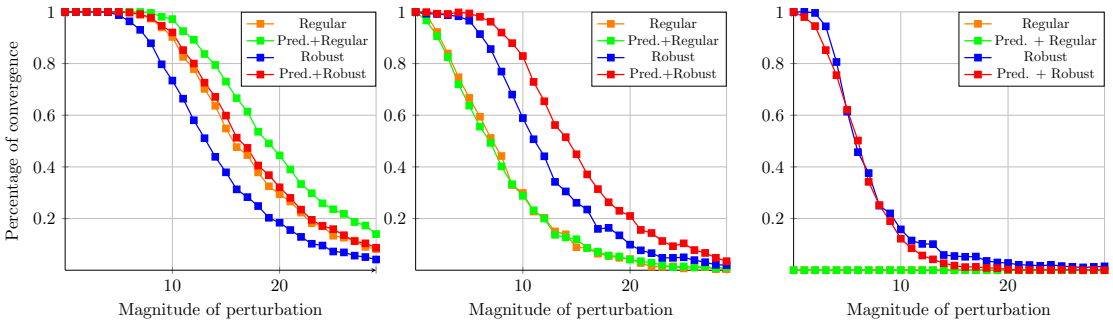
where r_i is the i-th normalized residual from (4), and $\rho(\cdot)$ is a robust function (at least C^0) that penalizes the largest residuals [7]. Note that for $\rho(\mathbf{r}_i(\mathbf{x})) = \mathbf{r}_i^2(\mathbf{x})$ with an unnormalized residual $\mathbf{r}_i$, we obtain the Least Squares cost function (4).

IRLS An important advantage of M-estimators is that they can be implemented using a simple Iteratively Reweighted Least Squares procedure, as performed in this software. The weights reflect the confidence of each datum and are computed as

$$w_i = \frac{1}{r_i}\frac{\partial \rho(r_i)}{\partial r_i}. \tag{7}$$

The procedure estimates $\mathbf{x}$ by solving a weighted Least Squares system, and reiterates until convergence. In any case there still exists a trade-off between efficiency and robustness. Relatively small interframe displacements or a suitable predictor should thus be applied, as shown bellow.

Figure 3 presents the frequency of convergence of the algorithm (over 1,000 cases for each magnitude of perturbation) for different setups. The "Regular" setup uses the default processing. The "Predictor + Regular" setup makes use of a 2D sliding window approach to initialize the position of the template on the current image before starting the optimization procedure. The "Robust" setup uses the strategies outlined in Section 2.1.5 to give weights to pixels in the image. Lastly, the "Predictor + Robust" setup uses both setups together. The same figure shows the frequency of convergence data for each setup with varying levels of occlusion. In the scenario with no occlusion (Fig. 3a), the best performance comes from the "Predictor + Regular" setup. In this case, the "Robust" setup suffers from a smaller basin of convergence. This occurs because it is always throwing away some piece of data, even if there is no occlusion. The importance of the "Robust" setup can already be seen in the 10% occlusion scenario (Fig. 3b). Indeed, nonrobust methods suffer because they cannot handle occlusions appropriately. The best setup is the "Predictor + Robust" in this case. For the last scenario of 20% occlusion (Fig. 3c), nonrobust methods fail completely, while the robust methods still work for relatively small perturbations.



(a) No occlusion (b) 10% occlusion (c) 20% occlusion

Figure 3: Frequency of convergence for different setups and occlusion levels.

Figure 4 shows the comparison of the average time needed by each setup to process an image when the occlusion is at the 10% level. Because each computer has different processing power, its data is normalized using the "Regular" setup value as the reference. It is clear that robust methods are slightly more computationally costly ($\approx$ 25%). This represents a trade-off to be decided by the end user.

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:

- the inter-frame displacement is relatively small;
- 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 mostly unoccluded.

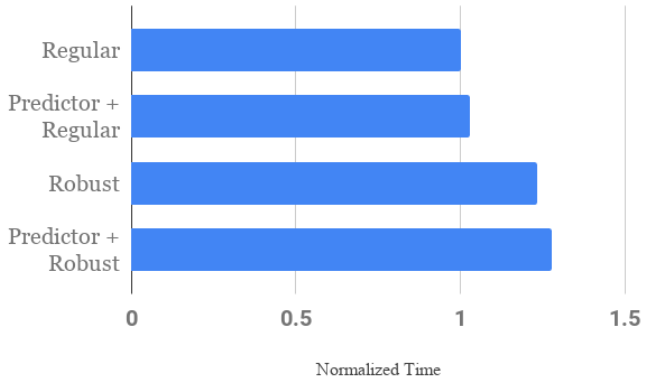


Figure 4: Average processing time for each setup using the "Regular" one as reference.

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.

3.1 Transformation variants

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 5 shows the Unified Modeling Language (UML) diagram for our software.

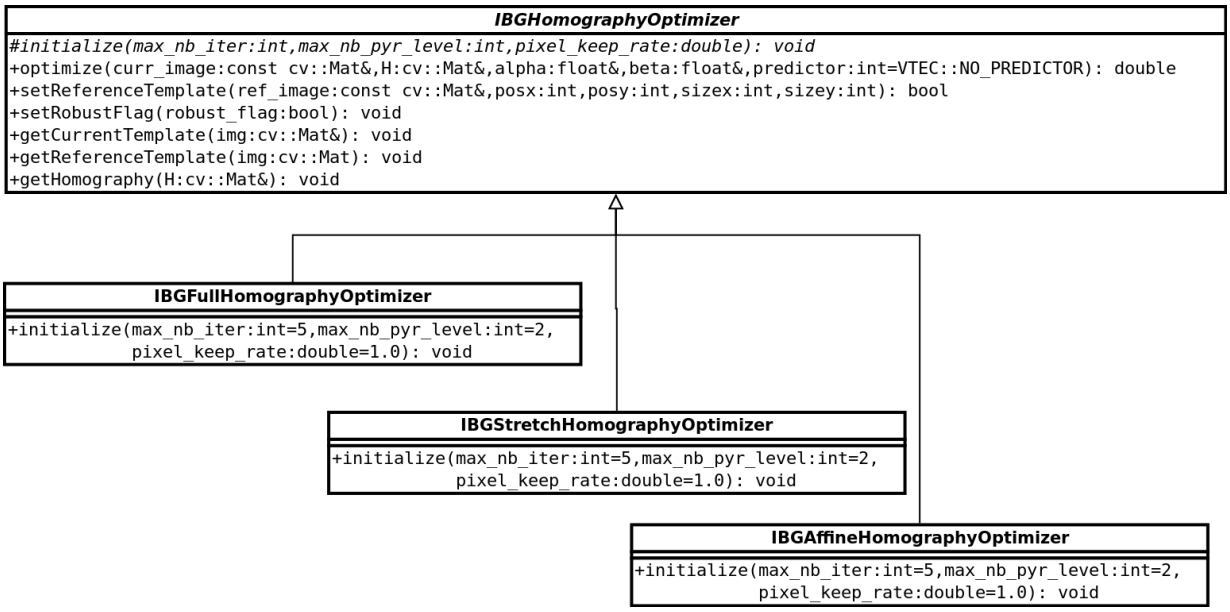


Figure 5: UML model of our homography optimization software.

3.2 Optional modes

The core algorithm jointly estimates the homography matrix and the gain and bias values, which represent the photogeometric parameters. Additionally, there are two modes that can be enabled independently, and that add functionalities to the core algorithm. These modes are presented below.

Predictor mode This mode adds an initialization step to the estimation procedure. The current homography matrix passed by the user as a parameter to the **optimize** function is refined using a sliding window approach. A small set of candidate homographies is generated by applying 2D translations to the current homography matrix. Each candidate is then scored using the ZNCC similarity measure. The candidate with the best score is then passed on to the nonlinear optimization step. To enable this mode, set the **predictor** parameter on the **optimize** function.

Robust mode This mode aims to increase the level of robustness to unknown occlusions within the homography optimization. It tries to identify occluded pixels by searching for outliers in the SSD residuals. Then, it eliminates the occluded pixels from the estimation process, and repeats this process iteratively. To enable this mode use the **setRobustFlag** function.

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 [8] 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

Our software has been tested on Ubuntu 16.04 and 18.04. Before you install it, 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
- OpenCV version 3.2.0 or later

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/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 */

std::ostringstream refFileNameStream;
refFileNameStream << directory << file_prefix <<
    std::setw(digit_width) << std::setfill('0') << 0 << ".pgm";

cv::Mat reference_image = cv::imread(refFileNameStream.str(),
    CV_LOAD_IMAGE_GRAYSCALE);
std::cout << refFileNameStream.str() << std::endl;

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 */
std::ostringstream currFileNameStream;
currFileNameStream.clear();
currFileNameStream << directory << file_prefix <<
    std::setw(digit_width) << std::setfill('0') << 5 << ".pgm";

std::cout << currFileNameStream.str() << std::endl;
cv::Mat current_image = cv::imread(currFileNameStream.str(),
    CV_LOAD_IMAGE_GRAYSCALE);
```

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 */
cv::Mat H(3, 3, CV_64F, cv::Scalar(0));
float alpha = 1.0;
float beta = 0.0;

/* Initialize Homography first guess */
H.at<double>(0, 0) = 1.0;
H.at<double>(0, 2) = BBOX_POS_X;
H.at<double>(1, 1) = 1.0;
H.at<double>(1, 2) = BBOX_POS_Y;
H.at<double>(2, 2) = 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.

```
cv::Mat current_template;
ibg_optimizer.getCurrentTemplate(current_template);

cv::Mat 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/ibg_tracker_with_predictor_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.

```
VTEC::IBGFullHomographyOptimizer optimizer;
optimizer.initialize(MAX_NB_ITERATION_PER_LEVEL, MAX_NB_PYR_LEVEL,
    SAMPLING_RATE);
optimizer.setReferenceTemplate(reference_image, BBOX_POS_X,
    BBOX_POS_Y, BBOX_SIZE_X, BBOX_SIZE_Y);
```

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

```
cv::Mat ref_template;
optimizer.getReferenceTemplate(ref_template);
cv::imwrite("./res/ref_template.pgm", ref_template);
```

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.

```
optimizer.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.

```
std::ostringstream fileNameStream;
fileNameStream << directory << file_prefix <<
    std::setw(digit_width) << std::setfill('0') << i << ".pgm";

std::cout << fileNameStream.str() << std::endl;

cv::Mat current_image = cv::imread(fileNameStream.str(),
    CV_LOAD_IMAGE_GRAYSCALE);

if (current_image.empty())
{
  std::cout << "End of image stream" << 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.

```
double score = optimizer.optimize(current_image, 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;

5.3 Robust homography-based visual tracking with occlusion handling

This use-case has a similar structure to the one in the previous section. In this case, an occlusion is introduced in the images after we have initialized the optimizer with an unoccluded reference template. To handle such partial occlusions, the robust mode is then enabled.

First, load and set the reference image normally.

```
std::ostringstream fileNameStream;
fileNameStream << directory << file_prefix << std::setw(digit_width)
               << std::setfill('0') << 0 << ".pgm";

cv::Mat reference_image =
    cv::imread(fileNameStream.str(), CV_LOAD_IMAGE_GRAYSCALE);

VTEC::IBGFullHomographyOptimizer optimizer;
optimizer.initialize(MAX_NB_ITERATION_PER_LEVEL, MAX_NB_PYR_LEVEL,
                     SAMPLING_RATE);
optimizer.setReferenceTemplate(reference_image, BBOX_POS_X,
   BBOX_POS_Y,
                                BBOX_SIZE_X, BBOX_SIZE_Y);
```

Put the optimizer in robust mode.

```
// Set Robust Flag on
optimizer.setRobustFlag(true);
```

Next, create the occlusion.

```
cv::Rect rect(0, BBOX_POS_Y, 500, BBOX_SIZE_X / 4);
```

For each image in the tracking sequence, add the occlusion to it and then call the optimize function as usual.

```
    // Apply occlusion object
    cv::rectangle(current_image, rect, cv::Scalar(0, 0, 0), -1);

    double score =
        optimizer.optimize(current_image, H, alpha, beta,
           VTEC::ZNCC_PREDICTOR);
```

This use-case outputs the same images as the previous use case. When using the robust optimization mode, it is highly recommended to also enable the predictor.

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 ibg_optimize_homography_example.cpp

```cpp
#include <homography_optimizer/ibg.h>
#include <iomanip>
#include <iostream>
#include <sstream>
#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 10

/* Number of levels in the pyramid, use more if you need more precision */
#define MAX_NB_PYR_LEVEL 4

/* 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 */

  std::ostringstream refFileNameStream;
```

```
38  refFileNameStream << directory << file_prefix << std::setw(digit_width) <<
    ↪  std::setfill('0') << 0 << ".pgm";
39
40  cv::Mat reference_image = cv::imread(refFileNameStream.str(),
    ↪  CV_LOAD_IMAGE_GRAYSCALE);
41  std::cout << refFileNameStream.str() << std::endl;
42
43  ibg_optimizer.setReferenceTemplate(reference_image, BBOX_POS_X, BBOX_POS_Y,
    ↪  BBOX_SIZE_X, BBOX_SIZE_Y);
44
45  /* Load current image */
46  std::ostringstream currFileNameStream;
47  currFileNameStream.clear();
48  currFileNameStream << directory << file_prefix << std::setw(digit_width) <<
    ↪  std::setfill('0') << 5 << ".pgm";
49
50  std::cout << currFileNameStream.str() << std::endl;
51  cv::Mat current_image = cv::imread(currFileNameStream.str(),
    ↪  CV_LOAD_IMAGE_GRAYSCALE);
52
53  /* Optimization parameters */
54  cv::Mat H(3, 3, CV_64F, cv::Scalar(0));
55  float alpha = 1.0;
56  float beta = 0.0;
57
58  /* Initialize Homography first guess */
59  H.at<double>(0, 0) = 1.0;
60  H.at<double>(0, 2) = BBOX_POS_X;
61  H.at<double>(1, 1) = 1.0;
62  H.at<double>(1, 2) = BBOX_POS_Y;
63  H.at<double>(2, 2) = 1.0;
64
65  std::cout << "Initial Homography: " << H << std::endl;
66
67  VTEC::drawResult(reference_image, H, 1.0, BBOX_SIZE_X, BBOX_SIZE_Y);
68
69  /* optimization stats */
70  double score;
71
72  /* optimize function */
73  score = ibg_optimizer.optimize(current_image, H, alpha, beta,
    ↪  VTEC::NO_PREDICTOR);
74
75  if (score == -1.0)
76  {
```

```
77      std::cout << "optimization failed" << std::endl;
78      return 0;
79    }
80    else
81    {
82      std::cout << "Score: " << score << std::endl;
83    }
84
85    /* Save the results */
86    VTEC::drawResult(current_image, H, score, BBOX_SIZE_X, BBOX_SIZE_Y);
87
88    std::cout << "Estimated Homography: " << H << std::endl;
89
90    cv::Mat current_template;
91    ibg_optimizer.getCurrentTemplate(current_template);
92
93    cv::Mat reference_template;
94    ibg_optimizer.getReferenceTemplate(reference_template);
95
96    /* Save images, annotated and the current template warped */
97    std::string annotaded_image_path, current_template_path,
      ↪  reference_template_path, reference_image_path;
98    annotaded_image_path = directory + "/res/annotated_image.pgm";
99    current_template_path = directory + "/res/warped_template.pgm";
100   reference_template_path = directory + "/res/reference_template.pgm";
101   reference_image_path = directory + "/res/reference_image.pgm";
102
103   cv::imwrite(reference_image_path, reference_image);
104   cv::imwrite(annotaded_image_path, current_image);
105   cv::imwrite(current_template_path, current_template);
106   cv::imwrite(reference_template_path, reference_template);
107
108   return 0;
109 }
```

B Listing of ibg_tracker_example.cpp

```
1  #include <homography_optimizer/ibg.h>
2  #include <homography_optimizer/aux.h>
3  #include <opencv2/highgui/highgui.hpp>
4  #include <iomanip>
5
6  /* Bounding box left-top position  (x,y) in the first image */
7  #define BBOX_POS_X 250
```

```
8  #define BBOX_POS_Y 200
9
10 /* Bounding box size (x, y) in the first image  */
11 #define BBOX_SIZE_X 200
12 #define BBOX_SIZE_Y 200
13
14 /* Maximum number of iterations per pyramid level. Lower means less computation
15  * effort */
16 #define MAX_NB_ITERATION_PER_LEVEL 5
17
18 /* Number of levels in the pyramid, use more if you need more precision */
19 #define MAX_NB_PYR_LEVEL 2
20
21 /* Sampling rate, 1.0 means 100% of points are used */
22 #define SAMPLING_RATE 1.0
23
24 int main(int argc, char** argv)
25 {
26   std::string directory = ".";
27   std::string file_prefix = "/seq/im";
28   int digit_width = 3;
29
30   int skip_image;
31
32   std::ostringstream fileNameStream;
33   fileNameStream << directory << file_prefix << std::setw(digit_width) <<
     ↪  std::setfill('0') << 0 << ".pgm";
34
35   cv::Mat reference_image = cv::imread(fileNameStream.str(),
     ↪  CV_LOAD_IMAGE_GRAYSCALE);
36
37   VTEC::IBGFullHomographyOptimizer optimizer;
38   optimizer.initialize(MAX_NB_ITERATION_PER_LEVEL, MAX_NB_PYR_LEVEL,
     ↪  SAMPLING_RATE);
39   optimizer.setReferenceTemplate(reference_image, BBOX_POS_X, BBOX_POS_Y,
     ↪  BBOX_SIZE_X, BBOX_SIZE_Y);
40
41   cv::Mat ref_template;
42   optimizer.getReferenceTemplate(ref_template);
43   cv::imwrite("./res/ref_template.pgm", ref_template);
44
45   cv::Mat H, cur_template;
46
47   optimizer.getHomography(H);
48
```

```
49    cv::Mat H_original = H;
50
51    int i;
52    float alpha = 1.0;
53    float beta = 0.0;
54
55    for (i = 0; i < 10; i += 3)
56    {
57      std::ostringstream fileNameStream;
58      fileNameStream << directory << file_prefix << std::setw(digit_width) <<
        ↪  std::setfill('0') << i << ".pgm";
59
60      std::cout << fileNameStream.str() << std::endl;
61
62      cv::Mat current_image = cv::imread(fileNameStream.str(),
        ↪  CV_LOAD_IMAGE_GRAYSCALE);
63
64      if (current_image.empty())
65      {
66        std::cout << "End of image stream" << std::endl;
67        break;
68      }
69
70      double score = optimizer.optimize(current_image, H, alpha, beta,
        ↪  VTEC::ZNCC_PREDICTOR);
71      std::cout << "RESULT" << std::endl;
72      std::cout << "Score: " << score << std::endl;
73
74      optimizer.getHomography(H);
75      std::cout << "Homography: " << H << std::endl;
76
77      VTEC::drawResult(current_image, H, score, BBOX_SIZE_X, BBOX_SIZE_Y);
78      cv::imwrite(directory + "/res/annotated_image_" + std::to_string(i) +
        ↪  ".pgm", current_image);
79
80      optimizer.getCurrentTemplate(cur_template);
81      cv::imwrite(directory + "/res/cur_template_" + std::to_string(i) + ".pgm",
        ↪  cur_template);
82    }
83  }
```

C Listing of ibg_tracker_robust_example.cpp

```cpp
#include <homography_optimizer/aux.h>
#include <homography_optimizer/ibg.h>
#include <iomanip>
#include <opencv2/highgui/highgui.hpp>

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

/* 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 3

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

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

  int skip_image;

  std::ostringstream fileNameStream;
  fileNameStream << directory << file_prefix << std::setw(digit_width)
                 << std::setfill('0') << 0 << ".pgm";

  cv::Mat reference_image =
      cv::imread(fileNameStream.str(), CV_LOAD_IMAGE_GRAYSCALE);

  VTEC::IBGFullHomographyOptimizer optimizer;
  optimizer.initialize(MAX_NB_ITERATION_PER_LEVEL, MAX_NB_PYR_LEVEL,
                       SAMPLING_RATE);
  optimizer.setReferenceTemplate(reference_image, BBOX_POS_X, BBOX_POS_Y,
                                 BBOX_SIZE_X, BBOX_SIZE_Y);

```

```
44    // Set Robust Flag on
45    optimizer.setRobustFlag(true);
46
47    cv::Mat ref_template;
48    optimizer.getReferenceTemplate(ref_template);
49    cv::imwrite("./res/ref_template.pgm", ref_template);
50
51    cv::Mat H, cur_template;
52
53    optimizer.getHomography(H);
54
55    cv::Mat H_original = H;
56
57    int i;
58    float alpha = 1.0;
59    float beta = 0.0;
60
61    // Create occlusion
62    cv::Rect rect(0, BBOX_POS_Y, 500, BBOX_SIZE_X / 4);
63
64    for (i = 0; i < 10; ++i) {
65      std::ostringstream fileNameStream;
66      fileNameStream << directory << file_prefix << std::setw(digit_width)
67                     << std::setfill('0') << i << ".pgm";
68
69      std::cout << fileNameStream.str() << std::endl;
70
71      cv::Mat current_image =
72          cv::imread(fileNameStream.str(), CV_LOAD_IMAGE_GRAYSCALE);
73
74      if (current_image.empty()) {
75        std::cout << "End of image stream" << std::endl;
76        break;
77      }
78
79      // Apply occlusion object
80      cv::rectangle(current_image, rect, cv::Scalar(0, 0, 0), -1);
81
82      double score =
83          optimizer.optimize(current_image, H, alpha, beta, VTEC::ZNCC_PREDICTOR);
84      std::cout << "RESULT" << std::endl;
85      std::cout << "Score: " << score << std::endl;
86
87      optimizer.getHomography(H);
88      std::cout << "Homography: " << H << std::endl;
```

```
89
90      VTEC::drawResult(current_image, H, score, BBOX_SIZE_X, BBOX_SIZE_Y);
91      cv::imwrite(directory + "/res/annotated_image_" + std::to_string(i) +
92                      ".pgm",
93                  current_image);
94
95      optimizer.getCurrentTemplate(cur_template);
96      cv::imwrite(directory + "/res/cur_template_" + std::to_string(i) + ".pgm",
97                  cur_template);
98    }
99  }
```

References

[1] G. Silveira, L. Mirisola, and P. Morin. Decoupled intensity-based nonmetric visual servo control. *IEEE Transactions on Control Systems Technology*, 2018.

[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] P. J. Huber. *Robust Statistics*. John Wiley & Sons, 1981.

[8] 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.