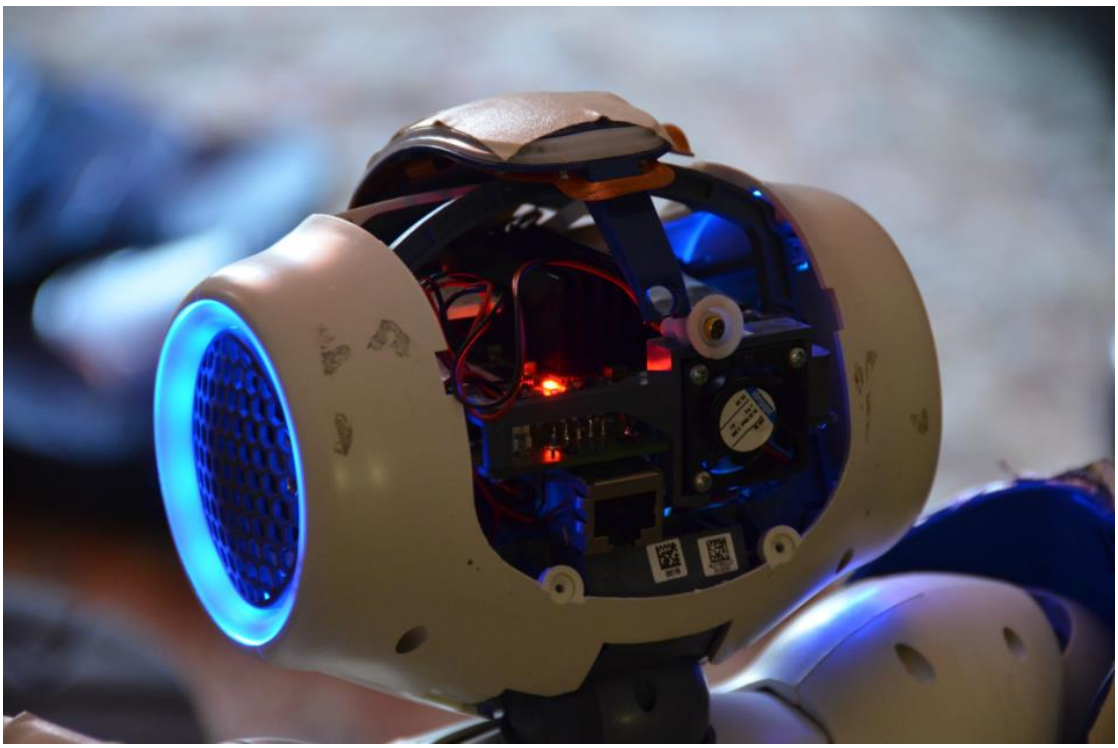
NAVIGATE

Nao Visual Gait And Trajectory Estimation



Supervisor: Dr. A. Visser

C. Kooijman	S. Laan	C. R. Verschoor	A. J. Wiggers
5743028	6036031	10017321	6036163
Chiel999@gmail.com	S.Laan@uva.nl	Verschoor@uva.nl	A.J.Wiggers@uva.nl

February 4, 2013
Project AI (6 EC)

MSc Artifical Intelligence
Faculty of Science
University of Amsterdam

Contents

1 Introduction

In the field of robotics, one of the main challenges is to estimate a robot's trajectory through the environment. One set of methods to tackle this problem focuses on both creating a map of its surroundings and determining the robots position in it, and is called Simultaneous Localisation And Mapping (SLAM). The process of estimating the trajectory is called odometry.

Some form of localisation is always needed for robots that move about freely, to correct errors in the execution of planned actions, which are inevitable in real-world situations. Localisation requires input from the robot's sensors, such as camera, laser range scanners, radar, sonar, or satellite navigation systems such as GPS (Easton, 1978) or GLONASS (Langley, 1997).

In this report we focus on monocular visual input only. This is useful because cameras are abundant, cheap, small, light, and require little energy. Cameras are passive sensors, which is useful when trying to avoid detection as well as in that they are not subject to interference from other sensors. Unlike satellite navigation, cameras work indoors given that there is enough light. On the other hand, video input is more difficult to process, as it is subject to lighting conditions and a single image does not provide any 3D information by itself. The benefit of estimation of this trajectory based on camera input over estimation based on modeling the camera's movement is that it is more robust to dynamic environments and motion error, especially in humanoid robots. If the camera's positions are estimated by triangulation of features in the perceived images, the environment can easily be reconstructed as a cloud of the 3D points that result from the triangulation. However, it is the trajectory estimation, and not reconstruction, that will be the main goal of this project.

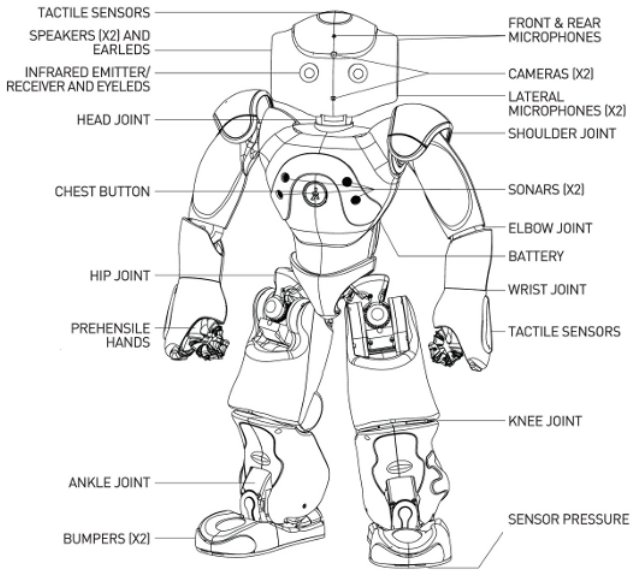


Figure 1: Schematic drawing of the Nao humanoid robot (Courtesy Aldebaran Robotics)

The application of visual odometry presented in this report is built for the humanoid robot Nao. As shown in figure 1, Nao was created by Aldebaran and is about 58 cm tall and has 25 degrees of freedom. To percieve its surroundings, two monocular cameras are located in its head. One of the cameras points in front of the robot and the other c points to the ground. The cameras of the Nao hardly overlap, therefore, stereo vision is not an option. additionally, Nao has

two ultrasound sensors in the chest that can be used for obstacle avoidance during navigation. The Nao comes with the NAOqi Application User Interface (API) containing the functionality for controlling the robot and pre-configured walking behaviours.

Section 2 gives an overview of related work that has been done and briefly arguments the approach proposed in this report. The basics of the theory this report relies on is explained in section 3. In section 4 the pipeline of our system is described stepwise. The framework that was built for the application of this report is presented in section 5. The discussion is covered in section 6, and conclusions are drawn in section 7.

2 Related Work

This section gives an overview of the related research that has been conducted regarding visual odometry for humanoid robots. Various visual odometry methods have already been investigated in the field. First the various related visual odometry applications are listed and briefly described and then an argumentation is given for the approach proposed in this report.

Overview

Oßwald et al. (2010, Section IV) describe a method for visual odometry on the Nao. Their approach requires the construction of a 2D feature map of the floor as prior knowledge. The Nao matches features found in the camera feed to the 2D feature map to perform localisation. As we do not assume a prior model of the environment in our approach, a 3D world model is required to calculate our position. Another option would be to use Image-to-Image transformations, but this would accumulate errors if multiple pairs of images need to be used and requires a decent overlap between the images, which is something that we can not always rely on, as the Nao has a very small field of view.

Esteban (2012) describes an approach for large scale city-size reconstruction and modeling using a monocular camera. The algorithm proposed can estimate the scaled translation of a camera optimally with as little as one correspondence between the 3D space and the 2D image. This approach also deals with the the problem of semantic modeling of large urban areas by merging information from different sources to reach a detailed building level description. The approach is more robust and less computationally expensive than alternative techniques.

Stereo-vision can also be used for visual odometry, and has the advantage that one time frame is enough to construct a 3D model, and the scale is can be calculated if the distance between the cameras is known. However, it requires stereo cameras that are not available in the Nao and many other devices such as laptops and smartphones. Stereo vision is similar to stereopsis in humans and other animals, but humans are also able to understand the world in three dimensions to a large degree when using only one eye. This suggests that monocular vision techniques may be useful in creating more robust or human-like stereo vision systems.

Motivation

We focus on keeping the program as light-weight and fast as possible, as it is intended to run in real time on a machine that also performs other tasks on the same processor, while also providing a stable, reliable estimation of the robots trajectory.

These observations lead us to the question we try to answer in this report: “Can we create a real-time monocular visual odometry system that reliably estimates the trajectory of a humanoid robot in a previously unknown environment?”

3 Theory

In this section, the basics of the theory behind the methods described in this report are explained. The work presented in this report is a continuation and extension of previous research regarding 3D modeling with a monocular camera by Esteban (2012) and builds on the Open Source Computer Vision Library (OpenCV) by Bradski and Kaehler (2008), which provides real-time computer vision implementations. This section aims to provide an overview of theory of the computer vision algorithms our method is based on.

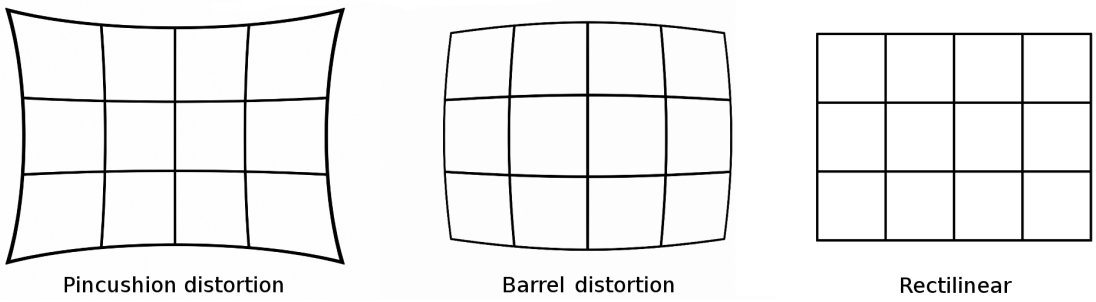


Figure 2: The two types of camera distortion: pincushion and barrel.

3.1 Camera Calibration

Raw images from the camera feed usually can not be used directly, as most are subject to some form of distortion. This distortion is different for each camera. There are two types of distortion, namely pincushion and barrel distortion (see figure 2). Combinations of these types are also possible. It is necessary to remove this distortion, otherwise, objects in the image may appear further or closer than they actually are. By estimating the distortion coefficients beforehand, retrieved images can be undistorted.

A method to find the camera parameters, proposed by Zhang (1999), uses a chessboard-like panel of which the dimensions are known, and is held up in front of the camera in different positions. The algorithm uses the correspondence between seen image points and object points, plus the knowledge that the panel is planar to infer the distortion coefficients k_1, k_2, p_1, p_2, k_3 and camera parameters f_x, f_y, c_x, c_y, forming the camera matrix:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

The focal length (in pixels) is represented in f_x and f_y in the x- and y-directions respectively. c_x and c_y denote the optimal camera center in pixels and should be near the center of the image. If the camera does not zoom, the focal length will not change and the calibration has to be performed only once. If zooming is possible, or multiple camera's are used, it is important to take the difference into account when performing methods based on this matrix.

3.2 Visual Odometry

In computer vision and probabilistic robotics, the term *visual odometry* refers to the estimation of the trajectory of a camera based on its perceived frames. The iterative process usually consists of these steps:

1. Some form of preprocessing (e.g. undistortion, scaling, colourspace conversion).
2. Extract features and keypoints from frame t.
3. Match found features with features from frame $t-1$.
4. Remove outliers from matches (for example, by using RANSAC)
5. Then, either
   - Update belief over possible states (as is the case for most Monte Carlo methods)
   - Find the state that minimizes some cost function based on reprojection error between frame t and $t+1$.

One of the main advantages of this method over estimating the trajectory based on motion modeling is that it is robust to motion noise. As an example, consider a wheeled robot that moves across rough terrain. Modeling the displacement is not an easy task for this environment, as wheels may slip or react different than to a flat surface. If the robot is mounted with a camera, the visual input can be used to estimate the displacement between frames. This does not require any modeling. Of course, noise over visual input will now be present instead of noise over motion. Which of these two will give higher performance therefore depends on the environment and the quality of the model.

3.3 Epipolar Geometry

In order to estimate the motion of the camera, relations between the different frames have to be identified. Epipolar geometry is employed to achieve this.

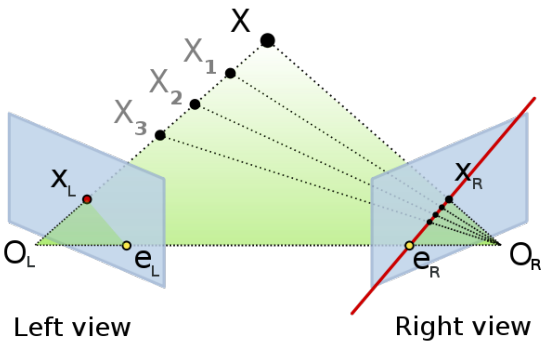


Figure 3: Visualisation of epipolar geometry (Courtesy Wikipedia)

One main concept in epipolar geometry is the fundamental matrix. The fundamental matrix relates matching points in two images (see figure 3.3). That is, if $\mathbf{x}_L$ and $\mathbf{x}_R$ are two observations of the same real world point $\mathbf{X}$, the fundamental matrix F imposes the following constraint:

$$\mathbf{x}_L^\top F \mathbf{x}_R = 0$$

Furthermore, if only one point is known, then $F\mathbf{x}$ specifies a line on which the matching point must lie. The fundamental matrix is of rank 2 and only defined up to scale.

Another related concept is the essential matrix, often denoted E. This matrix is related to the fundamental matrix with the following relation:

$$E = K^\top F K$$

Where K is the calibration matrix for the camera. Note that there is only one such matrix, as this is monocular vision. From this relation it can be seen that the fundamental matrix operates on uncalibrated cameras, while the essential matrix assumes cameras are already calibrated.

3.4 Estimating the fundamental matrix

Several methods exist for estimating the fundamental matrix from image correspondences. Some of the most popular ones will be described shortly.

3.4.1 Normalized 8-point algorithm

The simplest and most common method is the (normalized) 8-point algorithm (Longuet-Higgins, 1981; Hartley and Zisserman, 2000). Because the fundamental matrix has eight degrees of freedom, it can be estimated using eight point correspondences.

The 8-point algorithm finds an exact solution, if one exists, otherwise a least squares approximation is computed. An advantage of this method is that it can also cope with additional points. That is, if more points are provided, a more accurate estimate can be made.

It is common practice to normalize the matching keypoints before applying the 8-point algorithm. Hartley and Zisserman (2000, Chapter 11) argue that the normalization should not be considered optional. This normalization consists of translating and scaling the keypoint in such a way that they all lie round the centroid and have a mean distance of $\sqrt{2}$.

3.4.2 7-point algorithm

Because the fundamental matrix is of rank 2, it could be estimated using only eight points. However next to being of rank 2, it is defined up to scale, which means one more point correspondence can be dropped yielding a minimum of only seven points.

Using seven point correspondences, a polynomial of degree 3 can be created. This polynomial can then be solved analytically. This provides either one, or three solutions. If three solutions are returned, additional points are needed to determine which is correct.

3.4.3 5-point algorithm

The five point algorithm, introduced by Nistér (2004), reduces the number of required points even more, as the epipolar constraint is exploited even more. For each point correspondence, the eppolair costhatrnaint usmt ohl.d T is, $p'^\top E p = 0$, must hold for two matching points p and p'.

As Nister points out, this algorithm outperforms the 8-point algorithm on sideways motion, however, for forward motion, the 8-point algorithm is more robust. Because we predict that there will be more motion in the forward direction, the 8-point algorithm was chosen. Moreover, Nister suggests to use the 5-point algorithm in addition to the 8-point algorithm, meaning more computational costs.

3.4.4 RANSAC

Another popular method is the using random sample consensus (Fischler and Bolles, 1981; Hartley and Zisserman, 2000, Chapter 4.7.1) to obtain the fundamental matrix that best fits the given correspondences.

It first selects eight points to compute a fundamental matrix (using either 8-point or 7-point algorithm). Then, it checks how many of the remaining points agree with this solution model. The solution with the most support is chosen. As the selection of subsets is random, resulting fundamental matrices (and thus rotation and translation) will almost always differ for the same input.

3.5 Computing the projection matrix

From the essential matrix the projection matrix can be computed. The projection matrix P, specifies the rotation and translation necessary to move the first camera to the position of the second one. There are two approaches that are dominant in literature. Both are described briefly below.

3.5.1 Hartley and Zissermans method

Hartley and Zisserman (2000, Chapter 6) first specify a matrix W, that aids in computations:

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Then they show that there are two possible candidates for the rotation matrix and two possible translations, yielding a total of four possible solutions:

$$\begin{aligned} E &= U\Sigma V^\top && \text{(SVD)} \\ P_1 &= \left[UWV^\top | u_3\right] \\ P_2 &= \left[UW^\top V^\top | u_3\right] \\ P_3 &= \left[UWV^\top | -u_3\right] \\ P_4 &= \left[UW^\top V^\top | -u_3\right] \end{aligned}$$

Where U and $V^\top$ are the results of the singular value decomposition of E, and u_3 is the third column vector of U.

3.5.2 Horns Method

Another method that is frequently used for computing the projection matrix is the method of Horn (1990). It decomposes the essential matrix by employing the cofactors of the essential matrix.

The essential matrix is defined by $E = TR$, where T is the skew-symmetric matrix that satisfies $T\vec{v} = \vec{t} \times \vec{v}$ for translation vector $\vec{t}$ and any vector $\vec{v}$, and R the orientation. As $E = BR$, the following equality $EE^\top = TRR^\top T^\top = TT^\top$ must also hold. Now, T can be recovered independently from R by noting that $T^\top = -B$:

$$
\begin{aligned}
EE^\top &= -T^2 \\
T^2\vec{v} &= \vec{t}\times(\vec{t}\times\vec{v}) = \vec{t}(\vec{t}\cdot\vec{v}) - (\vec{t}\cdot\vec{t})\vec{v} \\
T^2 &= \vec{t}\,\vec{t}^{\,\top} - (\vec{t}\cdot\vec{t})I \\
\vec{t}\,\vec{t}^{\,\top} &= T^2 - (\vec{t}\cdot\vec{t})I \\
\vec{t}\,\vec{t}^{\,\top} &= \frac{1}{2}\mathrm{Trace}(EE^\top)I - EE^\top
\end{aligned}
$$

This gives us the matrix:

$$
\vec{t}\,\vec{t}^{\,\top} = \begin{bmatrix} t_1^2 & t_1t_2 & t_1t_3 \\ t_2t_1 & t_2^2 & t_2t_3 \\ t_3t_1 & t_3t_2 & t_3^2 \end{bmatrix}
$$

from which the translation vector can be derived by taking any column vector and dividing it by the square root of the diagonal element in that column. For numerical accuracy, the column with the largest diagonal value is picked. As there are two solutions for the square root of the diagonal element, this yields two solutions for the translation vector $\vec{t}$.

To recover the rotation, the matrix of cofactors of E is used. This matrix can be computed by taking the crossproduct of remaining columns for each column of E:

$$
\mathrm{Cofactors}(E) = \begin{bmatrix} e_2 \times e_3 & e_1 \times e_3 & e_1 \times e_2 \end{bmatrix}
$$

Next, because $E = TR$, $TE = T^2R = (\vec{t}\vec{t}^{\top})R - (\vec{t}\cdot\vec{t})R$. The matrix of cofactors can be rewritten in the form $\mathrm{Cofactors}(E) = ((\vec{t}\vec{t}^{\top})R)^{top}$. This gives us the following equation from which R can be derived, given $\vec{t}$:

$$
(\vec{t}\cdot\vec{t})R = \mathrm{Cofactors}(E)^\top - ET
$$

As we have found two solutions for t, this gives four candidate projection matrices consisting of either of two translation vectors and either of two rotation matrices.

3.6 Determining Rotation and Translation

As noted in the previous section, decomposition of the essential matrix gives four possible candidates for the projection matrix. This is to be expected, as there are multiple camera positions and orientations that will result in the same projection. Selection of the correct project matrix can be done by checking the depth of a perspective-projected point: If the z-coordinate of a point projected by the candidate projection matrix has a positive value, then this means that the point lies in front of the camera in 3D. In theory, only the correct projection matrix would result in positive depth for both camera positions. However, due to noisy images or mismatches during feature matching, some points will be projected incorrectly. Therefore, majority voting is applied to find the correct projection matrix. For every two matching points in two subsequent frames, a 3D position is calculated through triangulation. If the z-coordinate of this position is positive, it lies in front of the two cameras, and is therefore considered correct. The projection that results in the most correct 3D points is chosen as the true projection matrix.

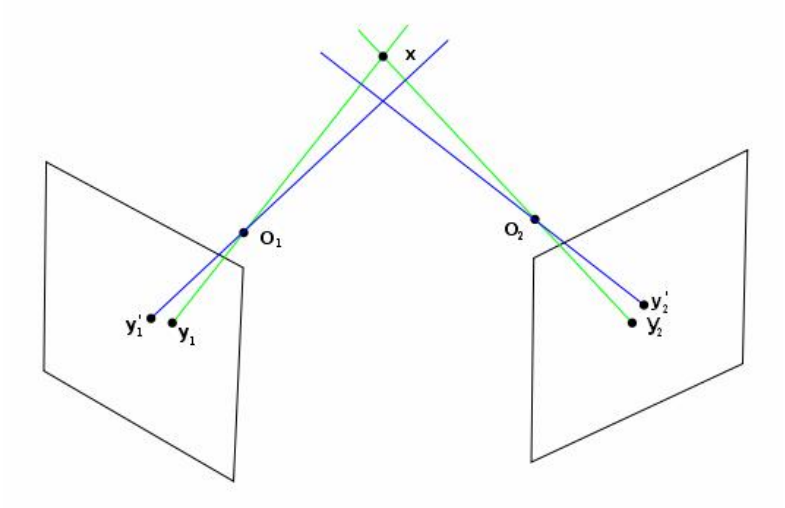


Figure 4: Some of the requirements for triangulation: Two camera frames with two points in those frames corresponding to a matching feature.

3.7 Triangulation

To find the 3D position of a feature in a set of images, at least two corresponding 2D points in subsequent frames, and the relative displacement and rotation of the camera(s) that took them are required. In the ideal situation, the vectors that represent the projection of the points intersect in the 3D point, as is the case in figure 4: here, points y_1 and y_2 are used to find the 3D position of a point x. The same can be done for points y_1' and y_2'. However, if any of these were to not intersect (as is often the case in real world applications due to noise in the images) a least-squares solution can be found for x which minimizes the distance between the vectors projected on each image plane through O. This process can be further improved by making the triangulation an iterative process, which reweighs parts of the transformation matrix. The process ends when the difference in weights per iteration falls below certain threshold. Although it is an iterative process (and therefore possibly slow), convergence is usually reached within 10 iterations (Hartley and Sturm, 1997), making it possible to use this method in real-time applications as well.

If rotation and translation are given by the transformation matrix that transforms camera coordinates from any given frame to the next, and we take the first frame as the reference frame, then the triangulation can be solved linearly by stating that a point u_1 in that frame is rotated and translated using the matrix:

$$P1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

whereas a point u_2 in the second frame is transformed by another matrix:

$$P2 = \begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & t_1 \\ R_{2,1} & R_{2,2} & R_{2,3} & t_2 \\ R_{3,1} & R_{3,2} & R_{3,3} & t_3 \end{bmatrix}$$

To find the point X in 3D that is the intersection of the vectors that represent the perspective projected image points, we construct matrices A and B:

$$A = \begin{bmatrix} -1 & 0 & u1_x \\ 0 & -1 & u1_y \\ u2_x \cdot P2_{3,1} - P2_{1,1} & u2_x \cdot P2_{3,2} - P2_{1,2} & u2_x \cdot P2_{3,3} - P2_{1,3} \\ u2_y \cdot P2_{3,1} - P2_{2,1} & u2_y \cdot P2_{3,2} - P2_{2,2} & u2_y \cdot P2_{3,3} - P2_{2,3} \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ u2_x \cdot P2(2,3) - t_1 \\ u2_y \cdot P2(2,3) - t_2 \end{bmatrix}$$

These are then used to find the least-squares solution for $AX = B$ using singular value decomposition.

3.8 Scale

A problem with Image-to-Image approaches is that in each image, the scale of seen objects differs from previous images. The impact of this becomes clear when multiple instances of the same object exist in a scene. When seen from a certain position and orientation, a small object may look the same as a large object seen from afar. The effect of this is that both compositions of the essential matrix triangulation will provide us with wrong positions for points in 3D.

To account for this scale difference, the relation between the scale in the very first image and any subsequent image can be determined by the relation between the current image points x and the 3D object points X in the image $mx_\alpha = \left[R|s\vec{t}\right] X_\alpha$:

$$\begin{bmatrix} mu_\alpha \\ mv_\alpha \\ m \end{bmatrix} = \begin{bmatrix} R_{1,1} & R_{2,1} & R_{3,1} & st_U \\ R_{2,1} & R_{2,2} & R_{2,3} & st_V \\ R_{3,1} & R_{3,2} & R_{3,3} & st_W \end{bmatrix} \begin{bmatrix} U_\alpha \\ V_\alpha \\ W_\alpha \\ 1 \end{bmatrix}$$

In this equation, m is the scaling factor needed to obtain image points. The factor we are trying to find, s, can be found by creating a set of equations from every correspondence by noting that:

$$mu_\alpha = r_1^\top X_\alpha + st_U \tag{1}$$
$$mv_\alpha = r_2^\top X_\alpha + st_V \tag{2}$$
$$m = r_3^\top X_\alpha + st_W \tag{3}$$

and that by substituting equation 1 in equations 2 and 3 a set of equations can be constructed in form $As = B$:

$$A = \begin{bmatrix} t_w u_\alpha - t_U \\ t_w v_\alpha - t_V \end{bmatrix}$$

$$B = \begin{bmatrix} (r_1^\top - r_3^\top u_\alpha) X_\alpha \\ (r_2^\top - r_3^\top v_\alpha) X_\alpha \end{bmatrix}$$

A least-squares solution for s can be found using singular value decomposition:

$$s = (A^\top A)^{-1} A^\top B$$

By calculating the local scale s for each frame, the translation can be scaled to correspond to the scale of the first frame. If the very first local scale is taken as reference, ratio between that and the current scale can be used to rescale the current translation. As soon as the metric 'local scale' can be expressed in terms of global scale units (e.g. meters, centimeters) every found 3D point and position can be expressed in these terms as well.

A second method involves combination of equations 2 and 3. Of course, a least-squares solution can also be found for only half of the values in matrix A, by disregarding values that are computed using either u or v. This would imply that u or v do not contribute to the scale. As the used platform can move in any given direction and rotates as well, it would seem illogical to disregard either. A disadvantage of doing this is that the error of the local scale will be the sum of errors from the set that uses u and the set that uses v. This is described in detail by Esteban (2012), who also dedicates a chapter to error propagation and selection of the optimal scale.

4 Pipeline

In this section, the pipeline of the proposed system is described stepwise. This system is a real-time monocular visual odometry system that estimates the trajectory of a humanoid robot in a previously unknown environment. All the previously described algorithms are implemented in this system. A schematic overview of the pipeline is shown in figure 5.

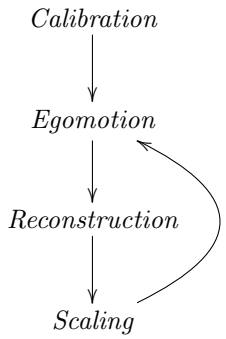


Figure 5: Schematic overview of the pipeline of the system

4.1 Calibration

Camera calibration is the process of estimating the intrinsic and extrinsic parameters of the camera. Images taken by the Nao robot's camera are subject to pincushion distortion (see figure 2). Calibration provides us with camera parameters that allow us to transform images in a way that removes the distortion. In figure 6, calibration is performed on two images taken on the Nao robot.

4.2 Egomotion

Egomotion or odometry is the process of estimating a camera's motion relative to a rigid scene. There are two main approaches for estimating the relative motion between two frames, namely, Image-to-Image and Image-to-World. Image-to-Image uses two images to find a homography and reconstruct their positions in 3D through triangulation. With Image-to-World we mean

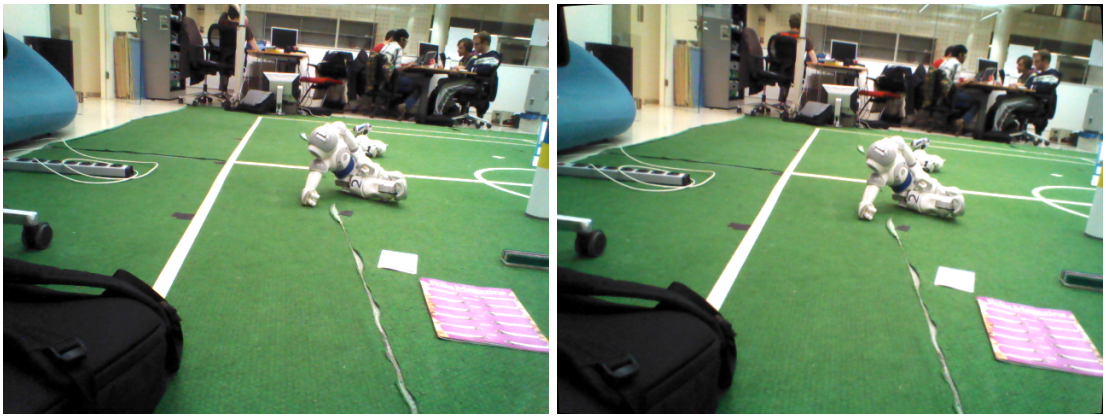


Figure 6: On the left side a distorted image and on the right side the undistorted image.

matching points in an image to their known positions in 3D to infer the position of the camera relative to the world coordinates. Although both methods were considered, the Image-to-World approach was never fully implemented.

4.2.1 Feature Extraction

In order to create a homography between two images (Image-to-Image), or match points of an image to a corresponding point in a 3D model (Image-to-World), we need to recognise and match these points in different images. The descriptions to match points (known as features or descriptors) need to be strict so that it does not match wrong pairs, but also tolerant so that it will still match points when conditions change. These conditions may include change of light, change of camera position causing a possible scaling or rotation, and motion blur. Finding descriptors also needs to be fast, so that it can be computed in runtime.

Three feature extractions methods are integrated into the framework, namely, Oriented FAST and Rotated BRIEF (ORB) by Rublee et al. (2011), Binary Robust Invariant Scalable Keypoints (BRISK) by Leutenegger et al. (2011) and Fast Retina Keypoints (FREAK) by Ortiz (2012).

ORB

ORB is a fast binary descriptor based on Binary Robust Independent Elementary Features (BRIEF) by Calonder et al. (2010), which is rotation invariant and resistant to noise. Furthermore, ORB builds on Features from Accelerated Segment Test (FAST) by Rosten and Drummond (2006). To the previous techniques, ORB adds efficient computation of oriented BRIEF features, analysis of variance and correlation of oriented BRIEF features and a learning method for decorrelating BRIEF features under rotational invariance, leading to better performance in nearest neighbour applications.

BRISK

BRISK relies on an easily configurable circular sampling pattern from which it computes brightness comparisons to form a binary descriptor string. The unique properties, rotation and scale invariance, of BRISK can be useful for a wide spectrum of applications, in particular for tasks with hard real-time constraints or limited computation power. BRISK finally offers the quality of high-end features in such time-demanding applications.

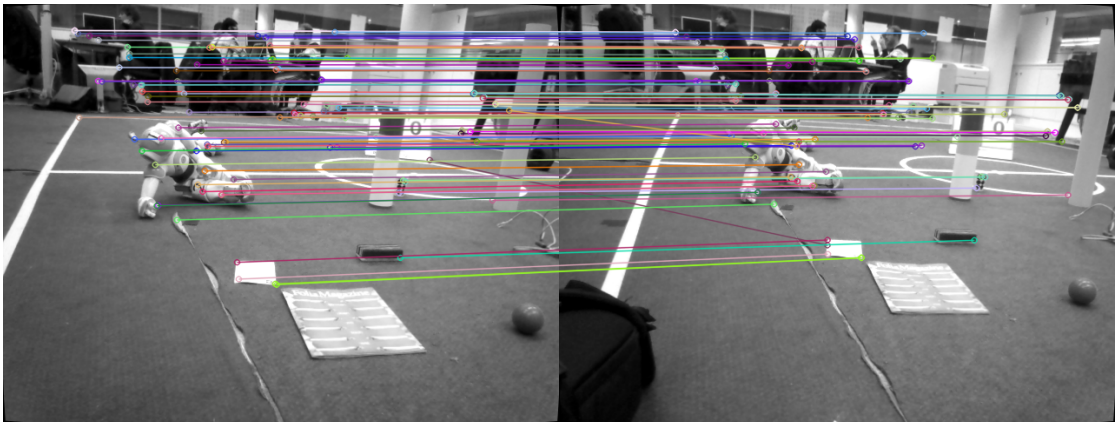


Figure 7: FLANN matching of BRISK features in two different frames.

FREAK

FREAK is a novel keypoint descriptor inspired by the human visual system and more precisely the retina. A cascade of binary strings is computed by efficiently comparing image intensities over a retinal sampling pattern, which is a circular pattern with having higher density of points near the center. As BRISK, FREAK is also rotation and scale invariant.

4.2.2 Feature Matching

For feature matching, the Fast Library for Approximate Nearest neighbour (FLANN) was employed (Muja and Lowe, 2009). As the name indicates this feature matcher uses nearest neighbour to determine the match of a keypoint (see figure 7). This means that while a match is always found, that match can be very different from the actual keypoint (for example when the real match is not in the set of potential matches), but it is faster than brute-force matching. Taking these properties into consideration, care must be taken when using the matches.

4.2.3 Image-to-Image

The Image-to-Image process starts with finding matches between the features in current and previous image. Then the matches have to be normalized for the normalized 8-point algorithm. Using this algorithm the fundamental matrix can be estimated. If RANSAC is used to determine the fundamental matrix, it can immediately provide the inliers. However, with the 8-point approach the outliers have to be rejected in a separate step. Once the fundamental matrix has been found and the inliers have been determined, the essential matrix can be computed. Using either Hartley and Zisserman (2000, Chapter 6) or Horn (1990) method, the four candidates for the projection matrix can be found. The final step consists of deciding which of the candidates is the right one. This is done via triangulation and a majority vote.

In short these are the steps for the image to image approach:

1. Match 2D with 2D descriptors
2. Normalize matches
3. Compute transformation matrices

4. Find fundamental matrix and use Random Sample Consensus (RANSAC) to reject outliers
5. Compute essential matrix
6. Estimation of projection matrix
7. Decide on the best transformation matrix from four candidates
8. Find the local scale
9. Rescaled transformation matrix

4.3 Triangulation

The previously described method estimates the motion of the camera and this pose estimation process yields the position of the different cameras in space. After obtaining pairs of feature points across frames, the 3D position of those points in space is estimated. To do this, at least two camera poses are needed (see figure 4).

In the pipeline, a simple approach is used, which assumes that the 3D point should lie on the shortest connecting line between two rays. The process of estimating the 3D position of the image matching feature is called triangulation (see section 3.7). Triangulation results into a 3D point cloud of the corresponding image matching features.

In this pipeline, we do not perform dense reconstruction as the point cloud is used for navigational purposes of the Nao and the current 3D feature points give enough information about the environment.

4.4 Scaling

After obtaining 3D points, the scale of the current frame can be estimated using the method specified in 3.8. This scale can be used to calculate the ratio between the current local scale and the scale in the very first 'reference' frame. If the ratio is known, 3D points can be rescaled to correspond to the points that are already present in the point cloud. Alongside this 3D point cloud, the feature descriptor and the coordinates of the image matching features are stored.

This is only necessary for Image-to-Image approaches, as in Image-to-World approaches, 2D points are matched to 3D points from the cloud; thus acquiring the scale of the already present points automatically.

In figure 8 a visualization of the resulting 3D point cloud over time is shown.

Figure 8: Visualization of the produced point cloud over time

4.5 Visual Odometry on Nao

The previous steps of the approach aid in calculation of rotation and translation of the robot (and, more specifically, its camera). The Nao robot can adjust its controls based on the new pose estimation provided by the system: The initial world coordinates and features are created from the first Image-to-Image data. As the robot moves around, the pose relative to the world can be calculated using the Image-to-Image approach, or, alternatively, an Image-to-World approach (PnP). To make sure that there are enough features in the world model, newly found features (and corresponding keypoints) can be added to the global set of descriptors, as the pose relative to world is known. The trajectory of the robot consists of the different camera positions relative to the world coordinates over time.

5 Framework

This section describes all components of the NAVIGATE integrated framework, which inherits the functionality explained in section 4. The NAOqi API is the reference project for writing applications for the humanoid robot Nao. In order to perform advanced tasks (e.g. sensor data processing and pose estimation) the NAVIGATE framework was created. The framework includes a generic data input class so any video device can be connected. Interfaces are used to connect the framework to specific video devices.

The NAVIGATE framework provides a keyboard interface for moving the Nao around and for recording data sets and calibrating the camera.

The framework is open source[1] and multiplatform. It has been tested on both Linux and Windows systems. It can be used on a computer with a recorded data set, with live data from the Nao, or can be cross-compiled to run on the Nao itself (although in that case, BRISK features can not be used, as the Nao uses OpenCV 2.3). It uses the OpenCV library (Bradski, 2000) for the implementation of several algorithms. Furthermore, the framework can visualise found 3D points using the Point Cloud Library (PCL) (Rusu and Cousins, 2011), but it is not needed when the software runs on the Nao.

6 Discussion

The approach that was used in this project will be analyzed, and a few discussion points will be addressed, along with possible future improvements.

6.1 Results

As the framework was not fully free from bugs as of the date of delivery, we were unable to run experiments and so can not provide any qualitative results with respect to the trajectory of the Nao based on its camera input. Therefore, we can not evaluate the performance of the visual odometry system.

We can, however, deliver some results regarding to runtime and number of features per descriptor for the test environment, and show that the developed application can in fact run in real-time (as it is only constrained by the speed at which images can be pulled from the Nao). The results for both ORB and BRISK are displayed in table 6.1, where mean and standard deviation of number of found features, along with time needed for evaluation are shown (for the same dataset of 21 images). Performance of FREAK could not be evaluated, as for this dataset,

[1] `https://github.com/aukejw/AI-Project--Visual-SLAM`

FREAK delivered too few features to be of use.

Descriptor	Mean	Standard deviation	Time needed
ORB	31.4	50.7	2.78
BRISK	67.5	24.5	3.33

We were also able to locate the faulty module. Candidates for the correct rotation and translation are based on decomposition of the essential matrix. This meant that the correct candidate would always have to be present, while this clearly was not always the case. We verified this by taking multiple snapshots from different known positions and orientations, so that the correct rotation and translation was known, and comparing it to every found candidate transformation matrix. As it turns out, the candidates for the rotation matrix were often illogical (such as matrices that indicate rotation around the z-axis, which would mean that the Nao had turned upside-down), or that they were too close to the identity matrix to be correct. This was true for both Hartley and Zisserman's method (where the translation is determined after the rotation) and Horns method (where the estimate of rotation is based on the translation).

This left only two candidate causes: Either the essential matrix is incorrect, or the decomposition of this matrix is at fault. As the fundamental matrix was computed using methods provided by OpenCV, and the essential matrix is directly based on this fundamental matrix, it could only be wrong if the camera matrix was incorrectly estimated. Although camera calibration was also done using methods provided by OpenCV, it is possible that the method we implemented is at fault. To find out if this is the cause of the problem, the implemented camera calibration methods will have to be tested on a system for which the camera matrix is already known.

6.2 Analysis

In our system, some methods were considered but eventually discarded. Here is a short overview of them and motivation for discarding them.

6.2.1 Stitching Nao Camera Images

Although the two cameras of the Nao cannot be used for stereo vision as they do not overlap, they could be used to enhance results. When images from both cameras are stitched together to form a bigger image, more features are likely to be found, which will probably result in a more accurate estimation of the essential matrix and hence the projection matrix. This does however increase the computational time, as stitching is a quite expensive operation. It was therefore not considered to use this method.

6.2.2 Scale Estimation

Another improvement can be made when estimating scale. Currently, the local scale is computed by a combination of equations which address translation in x- and y-direction. According to Esteban (2012), a system that moves in either x- or y-direction will benefit from using only the corresponding value. This is not the case for our platform, but it may still benefit from using a different method for scale estimation that does take the translation in both directions into account. A second improvement on scale estimation is the use of some local optimisation procedure that minimizes the reprojection error, as described in section 6.3.1.

6.2.3 Feature Selection

In this project, BRISK features were used due to matching speed and their robustness to rotation. Other descriptors that were considered are ORB and FREAK, both fast descriptor formats that are robust or invariant to rotation. These were both tried, however, both FREAK and ORB typically resulted in less features than BRISK, and were therefore not used. It may prove useful to use different descriptors for different environments, and ORBs invariance to rotation is a huge advantage when using a platform that can roam through the environment without being bound to direction constraints.

6.3 Future Work

This section describes the future work that can be done based on the findings during the creation of the pipeline proposed in this report. These are either extensions or improvements on the current proposed system.

6.3.1 Refinement through bundle adjustment

Both the camera parameters and the camera motion estimates can be refined during the process. A commonly used approach is the minimization of the reprojection error, which is defined as the sum of squared distances of reprojected 3D points and the 2D image points from which it was derived. An algorithm that aims to minimize this reprojection error in an iterative manner is the Levenberg-Marquardt algorithm (Levenberg, 1944; Marquardt, 1963; More, 1978). It has been shown that this method reduces reprojection error (Triggs et al., 2000). It was not used in this project due to the real-time constraint, but may prove to be useful in the future.

6.3.2 Cloud Stitching

Instead of saving keyframes with their features, a belief could be maintained over the whole world. This means that the different 3D locations of the features have to be integraed frame by frame. This can be done, but is quite expensive in terms of computational cost.

6.3.3 Image-to-World Approach

It was our intention to make the odometry less prone to incremental error by keeping a 3D world model of features. Instead of matching features in one frame to a previous frame, features are matched to a global world model in the form of a 3D point cloud. If a sufficient number of features occurs both in the image and in the point cloud, the rotation and translation can be estimated through algorithms that solve the Perspective-n-Point problem, such as EPnP by Lepetit et al. (2009): 'Given correspondences between points in the image and world coördinates, what is the position and orientation of the camera?'. Although this approach was considered for this project, it was never fully implemented.

If a feature occurs in at least two frames where displacement between camera positions is sufficient, triangulation can be used to find its position in 3D. These new features are then added to the cloud. An advantage of this method is that therm scale is implicitly estimated, which is not the case for frame-to-frame methods. On the other hand, to form the point cloud, it is necessary to keep track of a constantly growing point cloud and an even faster growing set of features from 2D images, needed for triangulation. A simple but naive approach is to restrict the depth of the history: If only points from the last n frames are remembered, the history will at least be finite. The problem with this approach is that it remains unsure when newfound 2D features are perceived again, and if we run out of 2D-to-2D correspondences, the 3D cloud

will become empty as well. A second approach is to only discard points from the history if the confidence in these points falls below a pre-set threshold, where confidence is determined by time since the point was last seen.

For this approach to work, there has to be an initial cloud of 3D points. In the first iteration, the Image-to-Image approach is employed to create the initial cloud and form the first transformation matrix. As soon as this cloud is formed, features in the 2D image can be matched against known features corresponding to the cloud's 3D points. When sufficient 2D-3D correspondences have been determined, rotation and translation can be estimated. Finally, points for which no 3D correspondence has been found can be triangulated if a match exists in the 2D history.

To summarise, the steps are:

1. Match 2D with 3D descriptors
2. Frame-to-frame, with RANSAC
3. Obtain rotation matrix from rotation vector
4. Triangulate any unknown points
5. Estimate scale, and rescale translation

6.3.4 Reconstruction

The reconstruction of the environment was not the goal of this project. Due to the real-time constraint, it was not feasible to create a 3D model of the environment alongside the trajectory estimation. It would however prove to be a useful addition to the framework if the environment could be visualized, for example for debugging purposes.

7 Conclusion

This report describes implementation and theory behind a system for trajectory estimation using monocular visual odometry for the Aldebaran Nao, a humanoid robot. The trajectory of the robot is estimated by using a frame-to-frame method, which uses displacement of features between two subsequent frames to find a rotation matrix and translation vector that indicate displacement of the camera. In this process, the essential matrix is computed, which is then decomposed into candidate rotation matrices and translation vectors. Alongside, 3D positions of corresponding keypoints in both frames are approximated using iterative triangulation. Based on the found points, the correct candidate rotation and translation are chosen. Several different methods are employed for decomposition of the essential matrix and triangulation.

The provided theory in this report gives ground for a system for trajectory estimation for the humanoid robot Nao. Although the performance could not be evaluated, we argue that the provided system can be used as a basis for a real-time application, that could perform fairly well on both trajectory estimation and in later stages, even 3D reconstruction.

As shown in table 6.1, the frame rate for BRISK and ORB features is 6.3 and 7.6 respectively. BRISK however finds more features per image, which suggests that it is the better choice for a robust visual odometry system.

References

G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

Gary Bradski and Adrian Kaehler. *Learning OpenCV*. O'Reilly Media Inc., 2008. URL `http://oreilly.com/catalog/9780596516130`.

M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. *Computer Vision–ECCV 2010*, pages 778–792, 2010.

RL Easton. The navigation technology program. *Navigation*, 25:107–112, 1978.

Isaac Esteban. *Large Scale Semantic 3D Modelling of the Urban Landscape*. PhD thesis, University of Amsterdam, December 2012.

M.A. Fischler and R.C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24 (6):381–395, 1981.

R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521623049, 2000.

Richard I. Hartley and Peter Sturm. Triangulation. *Computer Vision and Image Understanding*, 68(2):146 – 157, 1997. ISSN 1077-3142. doi: 10.1006/cviu.1997.0547. URL `http://www.sciencedirect.com/science/article/pii/S1077314297905476`.

Berthold K. P. Horn. Recovering baseline and orientation from essential matrix. *J. Optical Society of America*, 1990.

R.B. Langley. Glonass: review and update. *GPS world*, 8(7):46–51, 1997.

V. Lepetit, F. Moreno-Noguer, and P. Fua. Epnp: An accurate o (n) solution to the pnp problem. *International Journal of Computer Vision*, 81(2):155–166, 2009.

Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2548–2555, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-1-4577-1101-5. doi: 10.1109/ICCV.2011.6126542. URL `http://dx.doi.org/10.1109/ICCV.2011.6126542`.

K. Levenberg. A method for the solution of certain problems in least squares. *Quarterly of applied mathematics*, 2:164–168, 1944.

H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, January 1981. URL `http://www2.ece.ohio-state.edu/~aleix/Longuet-Higgins.pdf`.

D.W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11(2):431–441, 1963.

J. More. The levenberg-marquardt algorithm: implementation and theory. *Numerical analysis*, pages 105–116, 1978.

Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

D. Nistér. An efficient solution to the five-point relative pose problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(6):756–770, 2004.

Raphael Ortiz. Freak: Fast retina keypoint. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 510–517, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-1-4673-1226-4.

Stefan Oßwald, Armin Hornung, and Maren Bennewitz. Learning reliable and efficient navigation with a humanoid. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 2375–2380, May 2010. doi: 10.1109/ROBOT.2010.5509420. URL `http://hrl.informatik.uni-freiburg.de/papers/osswald10icra.pdf`.

E. Rosten and T. Drummond. Machine learning for high-speed corner detection. *Computer Vision–ECCV 2006*, pages 430–443, 2006.

Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R. Bradski. Orb: An efficient alternative to sift or surf. In Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc J. Van Gool, editors, *ICCV*, pages 2564–2571. IEEE, 2011. ISBN 978-1-4577-1101-5.

Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

Bill Triggs, Philip Mclauchlan, Richard Hartley, and Andrew Fitzgibbon. Bundle adjustment âĂŞ a modern synthesis. In *Vision Algorithms: Theory and Practice, LNCS*, pages 298–375. Springer Verlag, 2000.

Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 666–673. Ieee, 1999.