HAL
open science

Humanoid robot navigation: getting localization information from vision

Emilie Wirbel, Silvère Bonnabel, Arnaud de La Fortelle, Fabien Moutarde

To cite this version:

Emilie Wirbel, Silvère Bonnabel, Arnaud de La Fortelle, Fabien Moutarde. Humanoid robot navigation: getting localization information from vision. Journal of Intelligent Systems, 2014, 23 (2), pp.113-132. hal-00949738

HAL Id: hal-00949738

https://minesparis-psl.hal.science/hal-00949738

Submitted on 20 Feb 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Humanoid robot navigation: getting localization information from vision

Emilie Wirbel[*,**], Silvère Bonnabel[**], Arnaud de La Fortelle[**], and Fabien Moutarde[**]

[*]ewirbel@aldebaran-robotics.com,

A-Lab Aldebaran Robotics,

170 rue Raymond Losserand, 75015 Paris, France

[**]firstname.lastname@mines-paristech.fr,

MINES ParisTech, Centre for Robotics, 60 Bd St Michel 75006 Paris, France

January 28, 2014

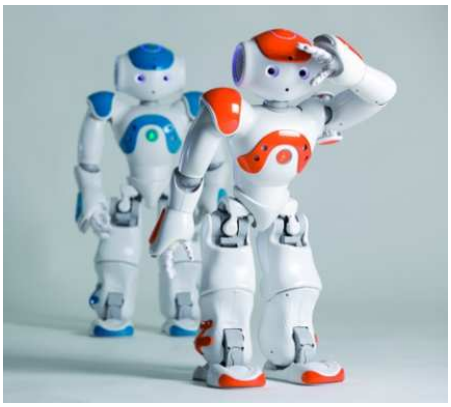
Abstract

In this article, we present our work to provide a navigation and localization system on a constrained humanoid platform, the NAO robot, without modifying the robot sensors. First we try to implement a simple and light version of classical monocular Simultaneous Localization and Mapping (SLAM) algorithms, while adapting to the CPU and camera quality, which turns out to be insufficient on the platform for the moment. From our work on keypoints tracking, we identify that some keypoints can be still accurately tracked at little cost, and use them to build a visual compass. This compass is then used to correct the robot walk, because it makes it possible to control the robot orientation accurately.

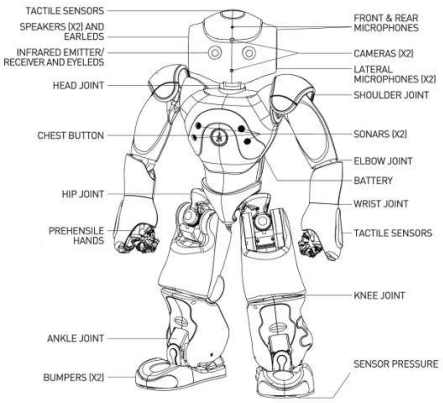
1 Introduction

This paper presents recent work that extends the results presented in Wirbel et al. (2013) and proposes new algorithms for the SLAM based on quantitative data.

1.1 Specificities of a humanoid platform: the NAO robot



(a) The NAO *NextGen* robot



(b) On board sensors and actuators

Figure 1: The humanoid platform NAO

All the following methods have been applied to the humanoid robot NAO. This robot is an affordable and flexible platform. It has 25 degrees of freedom, and each motor has a Magnetic Rotary Encoder (MRE) position sensor, which makes proprioception possible with a good precision (0.1°per MRE). Its sensing system include in particular two color cameras (see Figure 1). It is equipped with an on-board Intel ATOM Z530 1.6GHz CPU, and programmable in C++ and Python. A stable walk API is provided, but it is based only on the joint position sensors and the inertial unit (see Gouaillier et al. (2010)). It is affected by the feet slipping: the robot orientation is often not precise. This API makes it possible to control the robot position and speed.

Working on such a platform has undeniable advantages in terms of human-robot interaction, or more generally with the environment, but it also has specific constraints. The reduced computational power, limited field of view and resolution of the camera and unreliable odometry are very constraining factors. The aim here is to get some localization information without adding any sensor and still being able to run on line on the robot.

1.2 Related work

SLAM is a recurrent problem in robotics. It has been extensively covered, using metric, topological or hybrid approaches. However, the existing algorithms often have strong prerequisites in terms of sensors or computational power. Our goal here is to find a robust method on a highly constrained platform, which has not been specifically designed for navigation, such as the NAO robot.

Metric SLAM is the most common type of SLAM. Most rely on a metric sensor, such as a laser range sensor, a depth camera, a stereo pair, an array of sonars etc. Thrun (2002) presents a survey of some algorithms in that category: Kalman filter based, such as FastSLAM by Montemerlo et al. (2002), Expectation-Maximization (EM) algorithms by Thrun (2001), occupancy grids (originally from Elfes (1989)), etc.

Vision based metric SLAM rely on landmarks position estimation to provide the map, using mostly Kalman based approaches. For example Chekhlov et al. (2006) implement a SLAM based on Unscented Kalman Filter for position estimation and SIFT features for tracking: this method is designed to deal with high levels of blur and camera movements. However, these algorithms require to have a calibrated camera or a high resolution and frame rate, which is not available on the NAO: the extrinsic parameters of the camera vary from one robot to another. A VGA resolution is usually enough for most algorithms, but most existing algorithms use either a calibrated camera or a wider field of view than what is available on NAO. A higher resolution is available, but at a limited frame rate and at the cost of higher CPU usage. It is also worth noting that most hand-held visual SLAM implicitly rely on the hypothesis that the tracked keypoints are high above the ground and so of good quality and relatively easy to track. Among other related work, PTAM is very promising. The initial algorithm was described in Klein and Murray (2007), improved and optimized for a smartphone in Klein and Murray (2008), which is a setup quite close to NAO in terms of computing power and camera field of view, but does not seem to be fully reliable yet.

The previous methods use sparse keypoint information. It is also possible to use dense visual information to get qualitative localization. Matsumoto et al. (2000) implements navigation based on omni-directional view sequences. This kind of approach is heavier in terms of computational power, but more robust to blur, and less sensitive to texture quality. This is why it is also an interesting trail to explore.

Several approaches already exist to provide the NAO robot with a navigation system, but they all have strong prerequisites and / or require additional sensors. Some perform a metric localization and estimate the 6D pose of the robot by adding some metric sensor: Hornung et al. (2010) use a combination of laser range and inertial unit data, while Maier and Bennewitz (2012) use an Asus Xtion Pro Live depth sensor set on the top of the head. Both require a pre-built 3D map of the static environment, though Maier and Bennewitz (2012) also deals with dynamic obstacles. As far as vision is concerned, Maier et al. (2011) use sparse laser data to detect obstacles and classify them using vision. Osswald et al. (2010) estimate the pose using floor patches to perform reinforcement learning and reach a destination as fast as possible. It is worth noting that adding any kind of sensor on the robot (a laser

head or 3D camera) affects the robot balance and walk. Chang et al. (2011) propose a method for the fixed Robocup context, using only the regular sensors, but they rely on the specific field features.

The first aim of this article is to try out simple keypoints positions method, in order to be as light as possible in terms of computational power, which means in particular avoiding Kalman based approaches. In section 2, we describe a lightweight keypoints position estimation and try to adapt it to the NAO platform. We show that although we have good theoretical results, the application to the robot is difficult. Despite this, in section 3, we use some of our observations to derive a partial localization information, which consists in estimating the robot orientation, and test it both in simulation and on a real robot.

2 A metric visual SLAM algorithm

2.1 A keypoint based metric SLAM

2.1.1 Notations and model

Let x be the planar position of the robot, and θ_r its orientation in the world reference. For each keypoint indexed by i, we let p_i denote its position in the world reference. The robot axes are described in Figure 2.

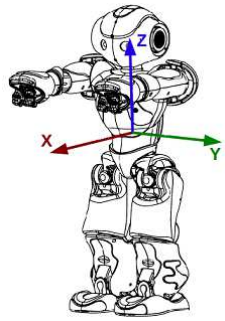


Figure 2: Axes definition on the robot

Letting ω denote the angular velocity of the robot, $u \in \mathbb{R}$ the value of its velocity expressed in m/s, R_θ the rotation of angle θ. If we assume that the robot velocity vector is collinear with the orientation

(a) Notations in the world reference

(b) Notations in the robot reference

Figure 3: Notations for the metric SLAM position estimation

axis $\mathbf{X}$ we end up with the following kinematic model for the motion of the robot:

$$\begin{aligned} \frac{d\theta_r}{dt} &= \omega \\ \frac{dx}{dt} &= uR_\theta \mathbf{X} \\ \frac{dp_i}{dt} &= 0 \end{aligned} \tag{1}$$

This model is quite simple, but corresponds to the existing walk API, which provides a speed control for translations and rotations. It is also possible to control the walk step by step, but since the focus of this work is not on the walk control, we have kept the high level simple control approximation.

In the robot reference frame $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$, the position of the keypoint i is the vector

$$z_i(t) = R_{-\theta(t)}(p_i - x(t))$$

The camera measurement being the direction (bearing) of the keypoint, for each i the vector

$$y_i(t) = \frac{1}{\|z_i(t)\|} z_i(t)$$

is assumed to be measured.

In the present paper, we propose a simplified two-step approach to the SLAM problem. First, the odometry is assumed to yield the true values of ω and u, and a filter is built to estimate each keypoint's position, and then, using those estimates globally, the knowledge of ω and u is refined (that is, the sensor's biases can be identified). The present section focuses on the first step, that is, estimating p_i from measurement y_i and perfectly known ω, u.

Our approach is based on the introduction of the vector $y_i^\perp = R_{\pi/2} y_i$. Letting $'$ denote the transposition of a vector, and writing that $y_i^\perp$ is orthogonal to y_i we have

$$\begin{aligned} 0 &= z_i'(t) y_i^\perp(t) \\ &= (R_{-\theta}(p_i - x(t)))' y_i^\perp(t) \\ &= (p_i - x(t))'(R_{\theta(t)} y_i^\perp(t)) \end{aligned} \tag{2}$$

Introducing the vector $\eta_i(t) = R_{\theta(t)} y_i^\perp(t)$ that is $y_i^\perp$ mapped into the world reference frame, we thus have

$$(p_i - x(t))'\eta(t) = 0 = \eta'(t)(p_i - x(t)) \tag{3}$$

The previous notations are illustrated on Figure 3: Figure 3a displays the notations in the absolute world reference and Figure 3b in the robot reference.

(2) simplifies all following computations by performing a reference change. Contrary to what is usually done, all observations are considered in the robot reference and not in an absolute reference.

Knowing at each time the previous relation (3) should be true, $p_i - x(t)$ and thus p_i (up to an initial translation $x(0)$) can be identified based on the observations over a time interval as the solution to a linear regression problem (the transformed outputs $\eta(t)$ being noisy). To tackle this problem on line, three main possibilities can be thought of.

2.1.2 Gradient observer

The problem can be tackled through least squares, trying to minimize at each step the following cost function:

$$C(t) = (\eta_i'(t)(\hat{p}_i(t) - \hat{x}(t)))^2 \tag{4}$$

To do so, we propose the (nonlinear) gradient descent observer (5), the estimation of x and θ being a pure integration of the sensor's outputs, and the estimation of p_i being $k_i > 0$ times the gradient of $C(t)$ with respect to p_i at the current iteration:

$$\begin{aligned} \frac{d\hat{\theta}}{dt} &= \omega \\ \frac{d\hat{x}}{dt} &= u R_{\hat{\theta}(t)} e_1 \\ \frac{d\hat{p}_i}{dt} &= -k_i \eta_i(t)'(\hat{p}_i(t) - \hat{x}_i(t))\eta_i(t) \qquad \forall i \end{aligned} \tag{5}$$

The gradient of the cost function with respect to p_i is:

$$\nabla C(t) = 2\eta_i'(t)(\hat{p}_i(t) - \hat{x}(t))\eta_i(t) \tag{6}$$

Let $e(t)$ denote the estimation error between the estimated keypoint position and the actual position in the robot reference (7), i.e.

$$e(t) = R_{\hat{\theta}_r(t)}(\hat{z}_i(t) - z_i(t)) \tag{7}$$

The following result allows to prove asymptotic convergence of the error to zero under some additional conditions.

Proposition 1.

$$\frac{d}{dt}\|e(t)\|^2 = -2k_i(\eta_i'(t)e(t))^2 \leq 0 \tag{8}$$

(8) proves that the estimation error is always non-increasing as the time derivative of its norm is non positive. Moreover, if there exists $T, \epsilon, \alpha > 0$ such that

$$\int_t^{t+T} \eta_i \eta_i'$$

is a matrix whose eigenvalues are lower bounded by ϵ, and upper bounded by α, the estimation error converges exponentially to zero (the input is said to be persistently exciting).

Proof. By construction:

$$\begin{aligned} \frac{d}{dt}\|e(t)\|^2 &= \frac{d}{dt}(e(t)'e(t)) \\ &= 2e'\frac{de}{dt} \end{aligned} \tag{9}$$

First we compute the error derivative:

$$\begin{aligned} \frac{de}{dt} &= \frac{d}{dt}(R_{\hat{\theta}_r}(\hat{z} - z)) \\ &= \frac{d}{dt}(R_{\hat{\theta}_r}(R_{-\hat{\theta}_r}(\hat{p} - \hat{x}) - R_{-\theta_r}(p - x))) \\ &= \frac{d}{dt}((\hat{p} - \hat{x}) - R_{\hat{\theta}_r - \theta_r}(p - x)) \\ &= \frac{d\hat{p}}{dt} - \frac{d\hat{x}}{dt} - \frac{d}{dt}(R_{\hat{\theta}_r - \theta_r}(p - x))) \end{aligned} \tag{10}$$

Now if we denote $\hat{\theta}_r(t) - \theta_r(t) = \tilde{\theta}(t)$:

$$\begin{aligned} \frac{d}{dt}(R_{\tilde{\theta}(t)}(p_i - x(t)))) = \frac{d}{dt}(\tilde{\theta}(t))\frac{d}{d\tilde{\theta}(t)}(R_{\tilde{\theta}(t)}(p_i - x(t))) \\ + R_{\tilde{\theta}(t)}\frac{dp}{dt} \\ - R_{\tilde{\theta}(t)}\frac{dx}{dt} \end{aligned}$$

We know that $\frac{d\hat{\theta}}{dt} = \frac{d\theta}{dt} = \omega$ by construction of the observer, so $\frac{d\tilde{\theta}(t)}{dt} = 0$ and also that $\frac{dx}{dt} = uR_\theta e_1$ and $\frac{dp}{dt} = 0$, as the observed point is considered fixed.

So

$$\begin{aligned}\frac{d}{dt}(R_{\tilde{\theta}(t)}(p-x))) &= -R_{\tilde{\theta}(t)}\frac{dx}{dt} \\ &= -R_{\tilde{\theta}(t)}uR_\theta e_1 \\ &= -uR_{\hat{\theta}}e_1\end{aligned}$$

So if we inject this in (10) we obtain

$$\frac{de}{dt} = \frac{d\hat{p}}{dt} - uR_{\hat{\theta}}e_1 + uR_{\hat{\theta}}e_1 = \frac{d\hat{p}}{dt}$$

So

$$\frac{de}{dt} = \frac{d\hat{p}}{dt} \tag{11}$$

So, using (9) and (11):

$$\begin{aligned}\frac{d}{dt}\|e\|^2 &= 2e'\frac{d\hat{p}}{dt} \\ &= -2ke'\eta(\hat{p}_i - \hat{x}_i)\eta'\end{aligned} \tag{12}$$

Now

$$(\hat{p}_i - \hat{x}_i)\eta' = (e + R_{\tilde{\theta}(t)}(p-x))\eta' \quad = e\eta' + eR_{\tilde{\theta}(t)}(p-x)\eta'$$

So if we use relation (3) we have

$$(\hat{p}_i - \hat{x}_i)\eta' = e\eta'$$

So finally if we inject this in (12)

$$\frac{d}{dt}\|e\|^2 = -2ke'\eta e\eta' \quad = -2k(\eta' e)^2$$

□

However in simulations, we see the convergence with such an observer can be very slow. This is because the instantaneous cost $C(t)$ is only an approximation of the true cost function viewed as an integral of $C(s)$ over all the past observations i.e. observed during the time interval $[0, t]$. For this reason we should consider some other possibilities.

2.1.3 Kalman filtering

The solution of this problem could be computed using a Kalman filter approach. Since we do not have any a priori knowledge of the keypoints positions, the initial covariance matrix would be $P_0 = \infty$.

Although this approach is perfectly valid, we will take advantage of the simplicity of the current problem and consider an even simpler solution, described in subsubsection 2.1.4. This way we avoid having to take care of numerical issues due to the initial jump of the covariance matrix from an infinite value to a finite one. We also avoid having to maintain a Riccati equation at each step.

Another similar solution would be to use a Recursive Least Squares (RLS) algorithm, with a forgetting factor $\lambda < 1$. The following simpler approach is closely related to RLS: it replaces the forgetting factor by a fixed size sliding time window.

2.1.4 Newton algorithm over a moving window

Due to the simplicity of the problem, and having noticed equation (2), we choose the following approach over Kalman or RLS filtering.

A somewhat similar approach to RLS consists in using a Newton algorithm to find the minimum of the cost function $C(s)$ integrated over a moving window. This has the merit to lead to simple equations and to be easily understandable. The cost at its minimum provides an indication of the quality of the keypoint estimation $\hat{p}_i$. We choose this simple alternative to Kalman filtering. Consider the averaged cost

$$Q(t,p) = \int_{t-T}^{t} (\eta_i \eta_i'(s)(p - x(s)))^2 ds \tag{13}$$

It is quadratic in p. Its gradient with respect to p merely writes

$$\nabla Q(t,p) = \int_{t-T}^{t} \eta_i \eta_i'(p_i - x(s)) ds = A(t)p - B(t)$$

The minimum is reached when the gradient is set equal to zero, that is

$$\hat{p}(t) = A^{-1}(t)B(t) \tag{14}$$

To be able to invert matrix $A(t)$, it has to be of full rank. By construction, it is possible iff the successive directions of the keypoint observations are not collinear. This means that the keypoint has been observed from two different viewpoints, which is the case in practice when the robot moves and the keypoint is not in the direction of the move.

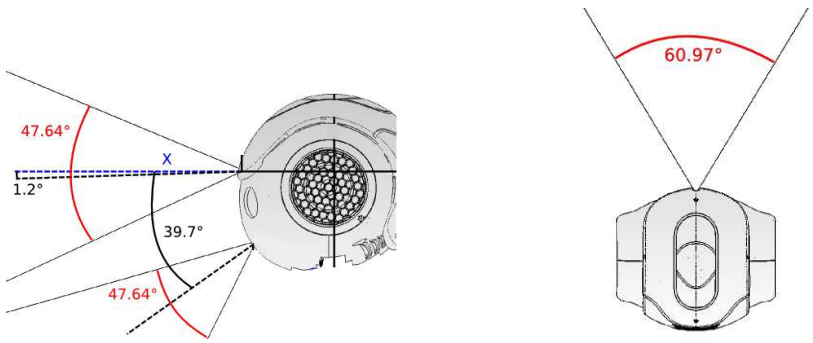
The convergence speed of this algorithm depends on the size of the sliding window T. It also depends on the determinant of matrix $A(t)$, which is easily interpreted: the triangulation will be better if we have observations which have very different bearings.

This algorithm can be interpreted as a kind of a triangulation. The cost function (13) can be interpreted as the sum of the distances from the estimated position to observation lines.

2.2 Adapting to the humanoid platform constraints

The previous algorithm is quite generic, but it requires that keypoints must be tracked robustly, and observed under different directions over time. This means that a few adaptations must be made on the NAO.

2.2.1 Images



(a) Side view (b) Top view

Figure 4: NAO robot head with camera fields of view

As shown in Figure 4, the on-board camera has a limited field of view. On the other side, the keypoints observations must be as different from each other as possible, which is obtained at best when the observed keypoints are observed in a perpendicular direction to the robot.

To compensate for this, we use the fact that the robot can move its head around and build semi-panoramic images. The robot moves its head to five different positions successively, without moving its base, stopping the head in order to avoid blur. Figure 5 shows an example of such a panorama. The different images are positioned using the camera position measurement in the robot reference, which is known with a precision under 1°thanks to the joint position sensors. This means that it is not required to run any image stitching algorithm, because this positioning is already satisfactory.

This makes it possible to have a larger field of view, and thus to track keypoints on the side of the robot, which are the ones where the triangulation works at best. However, it requires the robot to stop in order to have sharp and well positioned images. In practice, we have determined that the robot must stop and take a half panorama every 50cm.

Figure 5: Example of panoramic image

The robot head is not stabilized during the walk, which means the head is moving constantly up and down. Because of the exposure time of the camera, most of the images are blurred. There is also a rolling shutter effect: since the top part of the image is taken before the bottom part, the head movement causes a deformation of the image which seems “squeezed”. Because of these factors, it is necessary to stop the robot to get usable images.

The observer described in subsubsection 2.1.4 has been implemented in C++ to run on board on the robot.

2.2.2 Keypoints

The tracked keypoints are chosen for their computation cost and robustness. The keypoint detector is a multi-scale FAST (Features from Accelerated Segment test, (Rosten and Drummond, 2006)) detector because it is quick to compute and provides many points. The descriptor is Upright SURF or USURF, which is a variation of SURF (Bay et al., 2006) which is not rotation invariant and lighter to compute than regular SURF. The fact that the keypoints are not rotation invariant is an advantage here, because it reduces the risk of confusion between points which are similar after a rotation, for example a table border and a chair leg. This makes the tracking more robust and avoids false positives.

The keypoint tracking is performed by matching the keypoints descriptor using a nearest neighbor search with a KdTree to speed up the search. A keypoint is considered matched when its second nearest neighbor distance is sufficiently greater than the first nearest neighbor distance. This eliminates possible ambiguities in keypoints appearance: such keypoints are considered not sure enough to be retained.

The keypoint extraction and matching have been implemented in C++ using the OpenCV library (see OpenCV). Both FAST and USURF have proved robust enough to blur, so that the tracking error is usually less than a few pixels. The FAST detector has a pixel resolution, which means 0.09 degrees for a VGA resolution.

(a) Stops 0 to 4 (b) Stops 5 to 8

Figure 6: Robot stopping positions during an exploration run

2.3 Results and limitations

The previous algorithm has been tested on several runs in office environment (see Figure 5). During the runs, the robot walked around, stopping every 50cm to take a half panorama. The experiments were recorded in order to be re-playable, and filmed in order to compare the estimated keypoint positions to the real ones.

Figure 6 shows the different stopping positions of the robot during a run. The robot positions and orientation are marked with red lines. Figure 5 was taken at the first stop of this run.

Figure 7 shows a successful keypoint position estimation. The successive robot positions are marked with red arrows, where the arrow shows the direction of the robot body. The green lines show the observed keypoint directions. The pink cross corresponds to the estimated keypoint position.

The runs have shown that there are many keypoint observations that cannot be efficiently used for the observer. For each of these cases, a way to detect them and so ignore the keypoint has been implemented.

Parallel keypoint observations can be problematic. If the keypoint observations are exactly parallel, the matrix that has to be inverted is not of full rank. In practice, the observations are not exactly parallel, so the matrix would be invertible in theory. This leads to estimations of keypoints extremely far away, because of small measuring errors. To identify this case, we rely on the determinant of the matrix, which must be high enough. Figure 8 shows an example of such observations: the observations are nearly parallel, probably observing a keypoint very far away in the direction highlighted by the dashed blue line.

Because of matching errors or odometry errors, the triangulation may be of low quality, resulting in a triangulation with too much error. In that case, we can use the interpretation of the cost function as

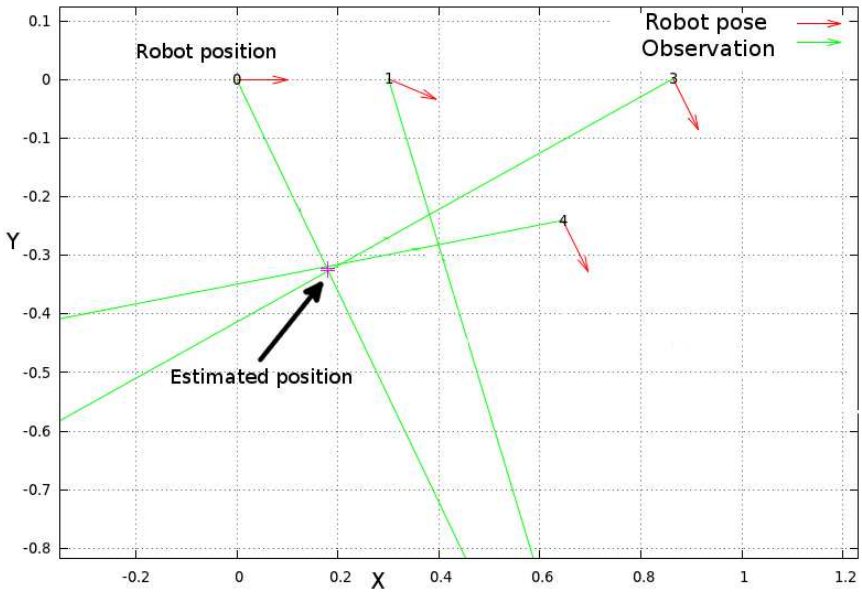


Figure 7: Example of correct triangulation from keypoints observations

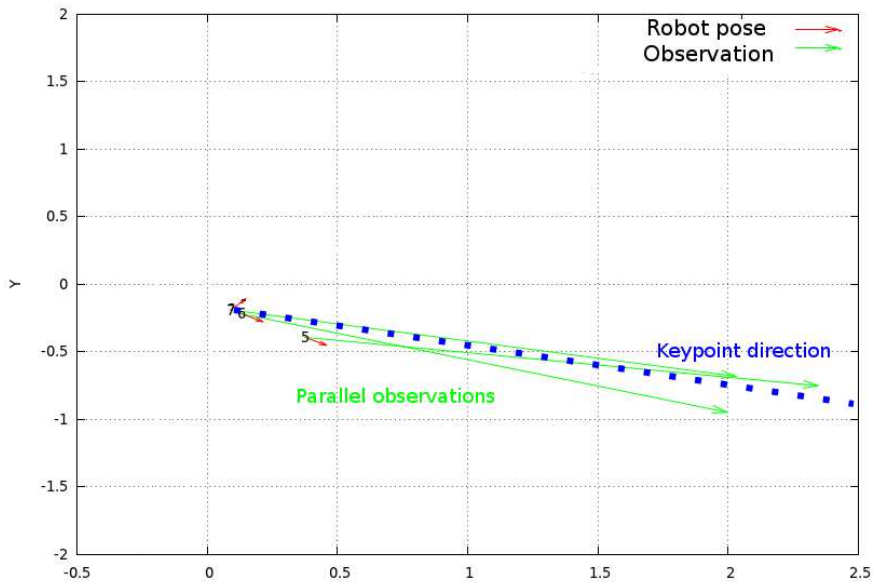


Figure 8: Parallel keypoints observations

the sum of distances from the estimated position to the observation to correlate its value with the size of the zone where the keypoints could be. Figure 9 shows an example of low quality triangulations: the estimated keypoint is 0.5m meters away from some observation lines. The triangulation zone is drawn with striped lines. A threshold on the cost value has been set experimentally to remove this low quality estimations.

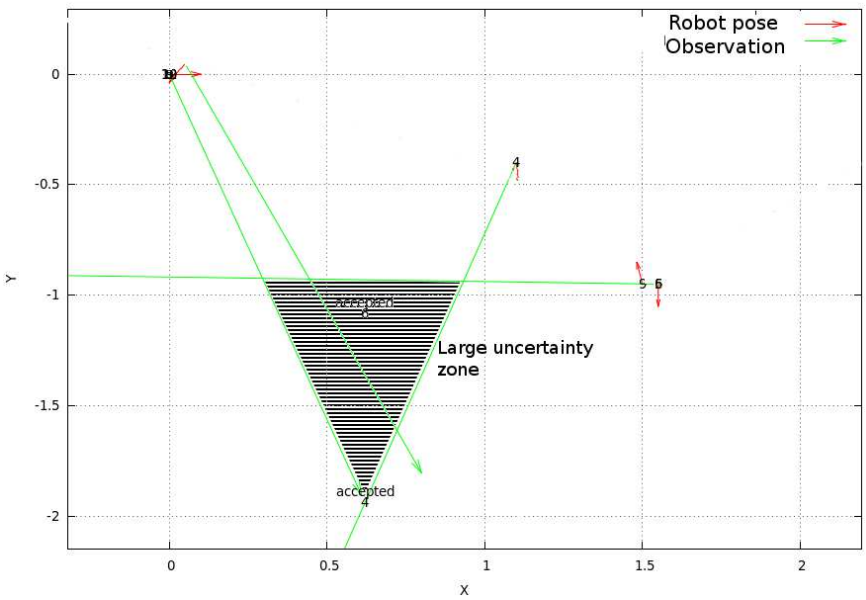


Figure 9: Low quality triangulation

Sometimes inconsistent observations can still result in an apparently satisfactory triangulation. One of the ways to detect this is to check that the estimated keypoint position is still consistent with the robot field of view. Figure 10 shows that the intersection between observations 0 and 4 is inconsistent because it is out of the robot field of view in position 4. This is the same for observations 0 and 8: the intersection is behind both positions. The observation directions have been highlighted with dashed blue lines, showing that the lines intersections should be invisible from at least one robot pose. To avoid this case, the algorithm checks that the estimation is consistent with every robot position in terms of field of view and distance (to be realistic, the distance must be between 20cm and 5m).

Figure 11 shows the influence of the different criteria on the keypoints estimation. The robot trajectory is drawn in red, with the robot positions in green, and the keypoint identifiers are put at the estimated positions. Note that the scale changes from one image to the other. Figure 11a presents the results without any criterion: the keypoints are very dense, but some estimations are too far away to be realistic. One keypoint is estimated 30m away for example, and most could not have been that far because of the room configuration. Figure 11b adds the quality criterion: some points are eliminated, and the distance values seem less absurd. The maximum distance is now around 6m. Figure 11c reduces the number of keypoints by adding the consistency criterion: it is visible that the number of unrealistic points has been reduced. For example, there are much less keypoints behind the first robot position. Finally, Figure 11d also adds the matrix determinant criterion. The distance to the keypoints is again reduced, which is consistent with the purpose of the criterion.

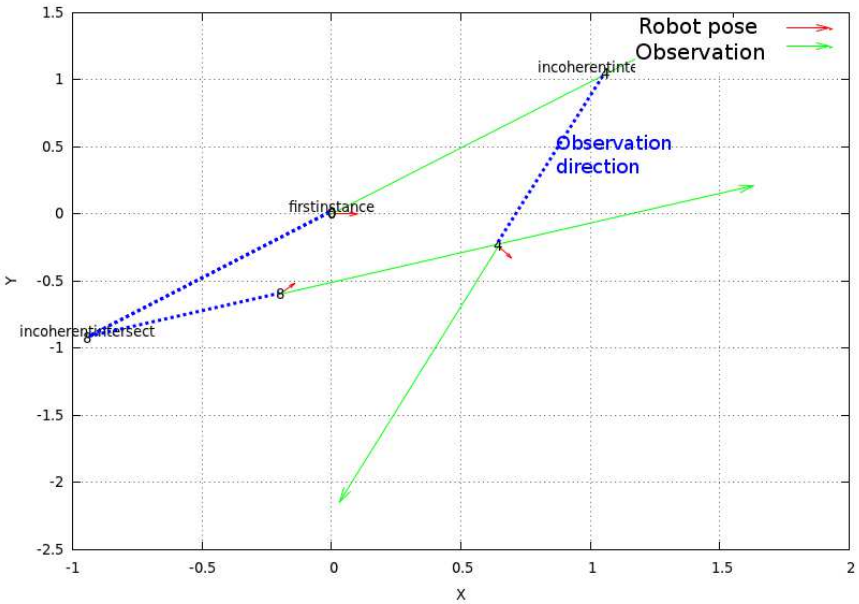


Figure 10: Inconsistent keypoints observations: intersection behind the robot

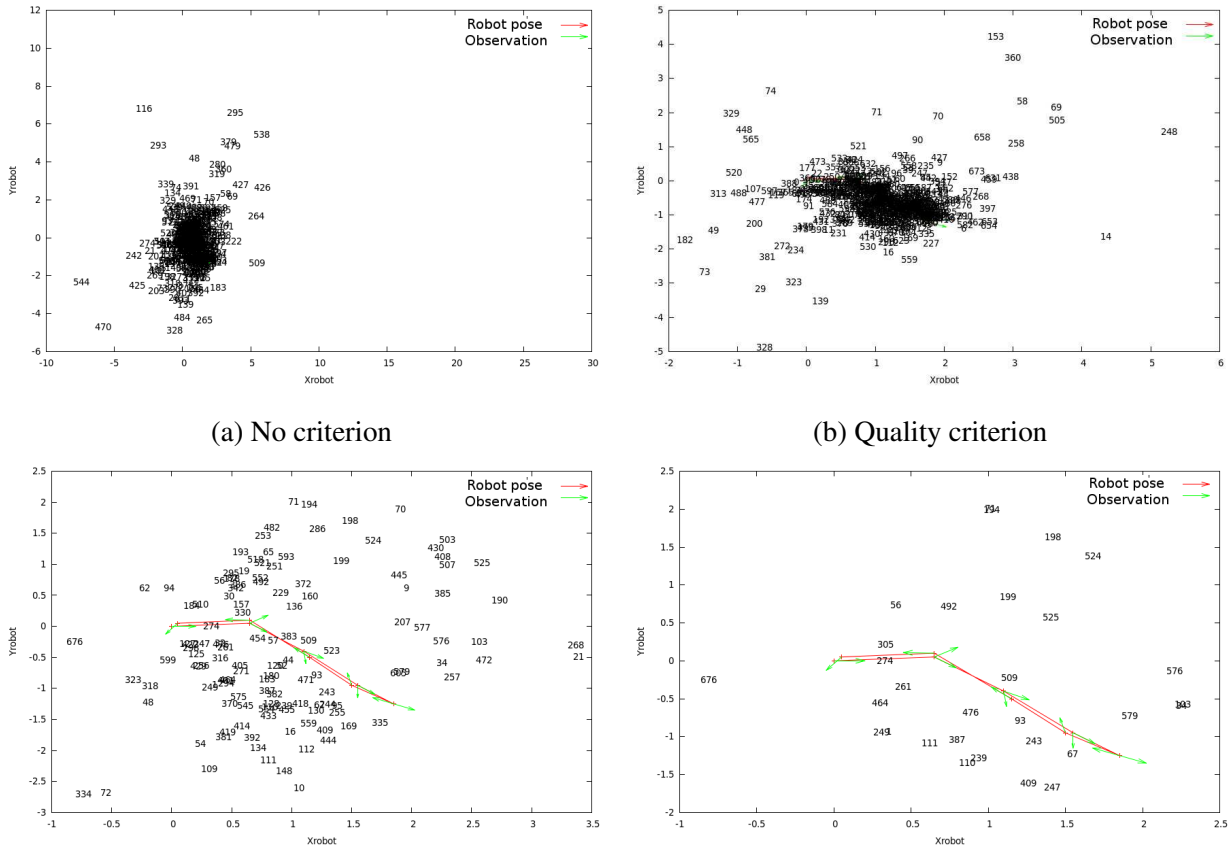
However, even if the number of unreliable estimations has been greatly reduced, an examination of the tracked keypoints has revealed that most of them are in fact false matches that happen to meet all of the previous criteria. The ones that effectively were correct matches were correctly estimated, but it was not possible to distinguish them from false matches.

This very high number of false matches comes from different causes. The first one is be perceptual aliasing: in the tested environment, there were a lot of similar chairs that were often mixed up. Possible solutions would be either a spatial check when matching keypoints and the use of color descriptors. However this would not solve all problematic cases.

Another difficulty comes from the size of the platform. At the height of the robot, parallax effects cause a large number of keypoints to appear or disappear from one position to the other. A way to solve this would be to take more half panoramas, but this implies more stops, which is problematic because the exploration is already very slow.

In fact, there were a lot of keypoints that were efficiently tracked. But they corresponded to keypoints that were very far away, or directly in front of the robot when it walked. These keypoints cannot be used because they do not satisfy our estimation hypothesis. In section 3, we try to take advantage of these points in another way.

In conclusion, even if simulation results seemed promising, and even if it was possible to meet some of the requirements of the algorithm, it was not enough to track reliable, high quality keypoints separately.



(a) No criterion

(b) Quality criterion

(c) Quality and consistency criteria

(d) Quality, consistency and determinant criteria

Figure 11: Effects of different criteria on the keypoint position estimations

3 A useful restriction: the visual compass

As explained in subsection 2.3, keypoints which are straight in front of the robot or which behave as if they were infinitely far away can be tracked. These observations cannot be used to approximate their positions, but they still can be useful.

3.1 From the metric SLAM to a compass

Keypoints that are infinitely far away (or behave as such) have a common characteristic: when the robot orientation changes, they shift accordingly in the image. It is the same when the robot performs pure rotation, without any hypothesis on the keypoint actual position. Ideally, when the orientation changes, all keypoints will shift of the same number of pixels in the image. This shift depends on the image resolution and camera field of view, and can be known in advance. Here we make the assumption that the camera intrinsic parameters do not deform the image, and that we can simply use a proportional

rule to convert a pixel shift into an angular shift. This assumption is only a simplification, so a more complex model could be used, and it is met in practice on the camera of NAO.

3.1.1 Observation of the deviation angle

Ideally, if all keypoints were tracked perfectly, and satisfying the desired hypothesis, only one pair of tracked point would be required to get the robot orientation. However, this is not the case because of possible keypoints mismatches, imprecisions in the keypoints position due to the image resolution. The assumption that the robot makes only pure rotations is not true, because the head is going up and down during the walk, and the robot torso is not exactly straight. This implies some pitch around the Y axis and roll around the X axis which added to the yaw deviation around the Z axis (see Figure 2 for the axes definitions). When the robot is going forward, because the tracked keypoints are not actually infinitely far away, the zooming or dezooming effect will add to the shifting, and our assumptions will be less and less valid.

To compensate for this effect, a possible solution is to rely on all tracked keypoints to get the most appropriate global rotation. A way to this is to use RANdom SAmple Consensus (RANSAC) (or any variation such as PROSac). A model with two pairs of matched keypoints is used to deduce the yaw, pitch and roll angles.

To make the model computations easier, we will use the complex representation of the keypoints position. Let z_1 be the position of the first keypoint in the reference image, and z'_1 its position in the current image. z_2 and z'_2 are defined in a similar fashion for the second keypoint. We define the following variables

$$\begin{aligned} \theta &= \arg\left(\frac{z'_1 - z'_2}{z_1 - z_2}\right) \\ o &= z'_1 - z_1 e^{i\theta} \end{aligned} \tag{15}$$

The angles can then be obtained by the following equations, if c is the image center coordinates:

$$\begin{aligned} \omega_x &= \theta \\ \omega_y &= \Im(o - (1 - e^{i\theta})c) \\ \omega_z &= \Re(o - (1 - e^{i\theta})c) \end{aligned}$$

It is worth noticing that the model (15) is not usable for every couple of matched point. The model assumes that this pair of points is consistent with the model. This is not the case for example if the keypoints are mismatched. However this false computations will be eliminated by the RANSAC, because they cannot have more inliers than a model based on valid points.

To compute the inliers for one particular model, we have to define a distance from one point to the other. For a point pair (z, z'), we compute the theoretical image of z:

$$\tilde{z} = e^{i\theta} z + o$$

The distance is then:

$$d = \|\tilde{z} - z'\| \tag{16}$$

We can then use the model computations from (15) and the distance computation from (16) to perform the algorithm. The number of inliers from the final best model can be used as an indication of how reliable the computation is. In particular, this can be used when the assumption that keypoints are infinitely far away becomes dubious, as in subsubsection 3.1.2.

The keypoints used are multi-scale FAST detector as before, but the chosen descriptor was the ORB descriptor (Oriented BRIEF, see Rublee et al. (2011)) for speed and computational cost reasons. This descriptor is a vector of 256 binary values, which are quicker to extract and to compare than the usual SURF descriptors, with only on a slight difference in robustness.

In practice, the camera angular resolution is $7 * 10^{-3}$ radians per pixel for a 160x120 resolution (which is the one used here). This means that if we have an uncertainty of 5 pixels, which is a reasonable upper bound for the uncertainty of the keypoint detection, this results in an uncertainty of about 2 degrees maximum, which is a considerable improvement from the uncorrected walk (see for example Figure 16 for a comparison).

The robustness and precision of this method is linked to the keypoint density. In practice, most indoors or outdoors environment have enough keypoints to provide a reliable heading estimation.

3.1.2 Control of the robot movement

The principle of our visual compass is to use this information to control the robot orientation. Let ω be the angle to rotate. Before rotating, the robot first turns its head towards a reference direction. This direction is situated at the angle position $-\frac{\omega}{2}$ in the initial robot reference, so that any rotation within $[-\pi, \pi]$ is possible. The robot then maintains its head in that direction and rotates its body towards the final orientation. The head target position is 0, and the base target position is $\frac{\omega}{2}$, in the direction reference. Let α be the head angle in the fixed reference and θ be the body angle.Their initial values are 0 and $-\frac{\omega}{2}$ respectively. The robot is controlled with its body rotation speed $\dot{\theta}$ and with its head speed in the robot reference, which is $\dot{\alpha} - \dot{\theta}$.

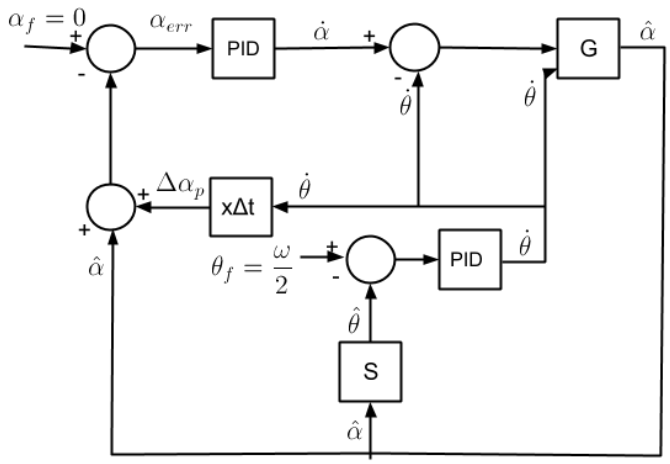


Figure 12: PID controllers for the head and robot body

The control is done with two PID controllers, described in Figure 12. The observed variable is the current angular deviation of the head in the fixed reference, $\hat{\alpha}$. We deduce the current deviation of the robot body using the internal position sensors of the robot, S. The desired robot body speed is then computed. It is used to predict the head movement due the body movement $\Delta\alpha_p = \dot{\theta}\Delta t$, which is then added to the current observed angle for the error computation. This step has been added to make the head tracking more robust for high rotation speeds.

For translations, it is possible to make the robot walk a straight line by simply adding a forward speed $\dot{x}$. When the estimation starts being less reliable (see subsubsection 3.1.1), the robot stops going forward, aligns itself in the desired direction and renews its reference image. This makes the reference reliable again, and the robot can keep going forward.

3.2 Simulation results on Webots

To obtain quantitative results with a reliable ground truth measurement, we have used Webots simulator from Cyberbotics. This simulator provides physics simulation along with various sensors simulation, including cameras. A simulated world has been built to test the visual compass (see Figure 13) using the simulator as a ground truth. This simulator emulates a real robot and reproduces the architecture of the robot operating system, NAOqi. The simulator differs from the real world in the odometry estimation which is slightly worse in the simulator (because the robot is slipping more) and the images are more stable (because there is no motion blur or lighting conditions changes). However this is an interesting validation step, and makes it possible to get repeatable and measurable experiments.

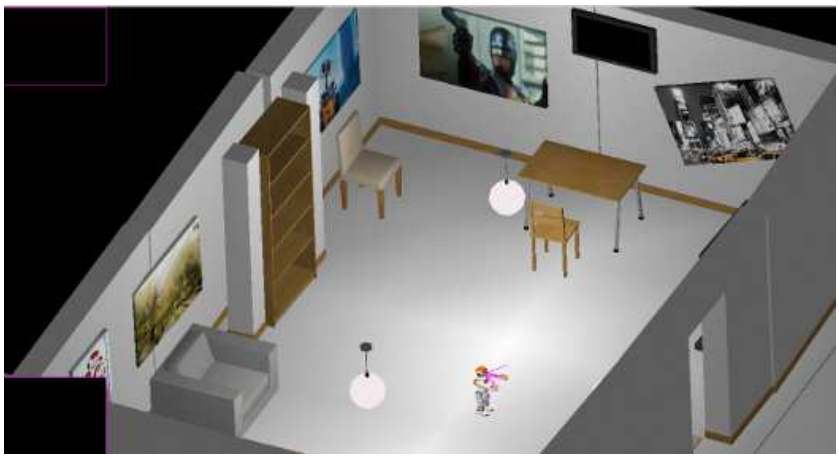


Figure 13: The simulated environment (using Webots)

Two different simulated experiments have been run. During each experiment, we record the estimated angle from the compass, the ground truth position of the robot and the estimated position of the robot by the odometry. The odometry and compass estimation are reset to the ground truth between each new target movement.

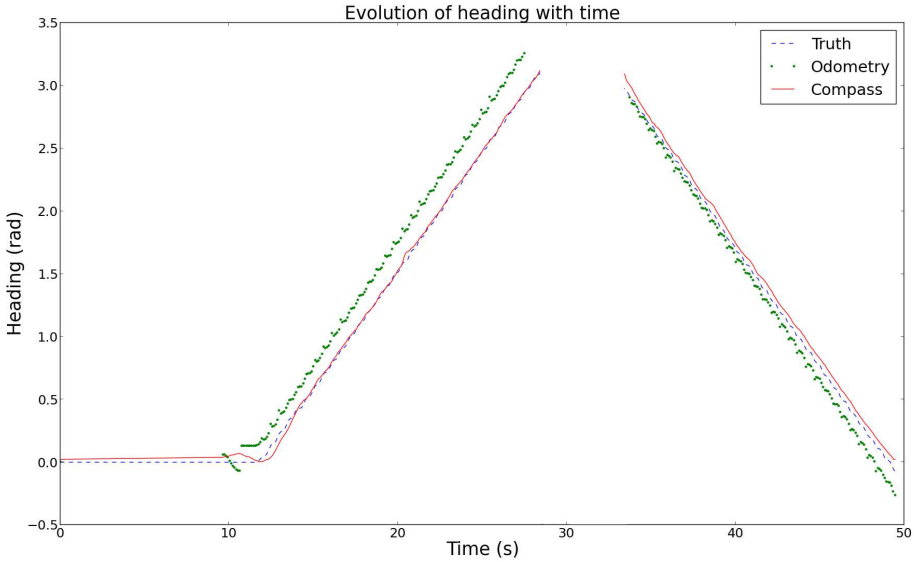


Figure 14: Comparison of the robot angle estimations during pure rotations (two rotations of 20s each)

The first experiment makes the robot perform two consecutive pure rotations, of an angle π then $-\pi$ (each lasting around 20 seconds in the simulator). Figure 14 shows a comparison of the different estimations of the robot base angle. The ground truth angle is drawn in dashed blue line, the compass angle is drawn in full red line, and the odometry angle is drawn in dotted green line. It is visible that

the odometry measurement drifts steadily during the movement, while the compass estimation stays close to the ground truth.

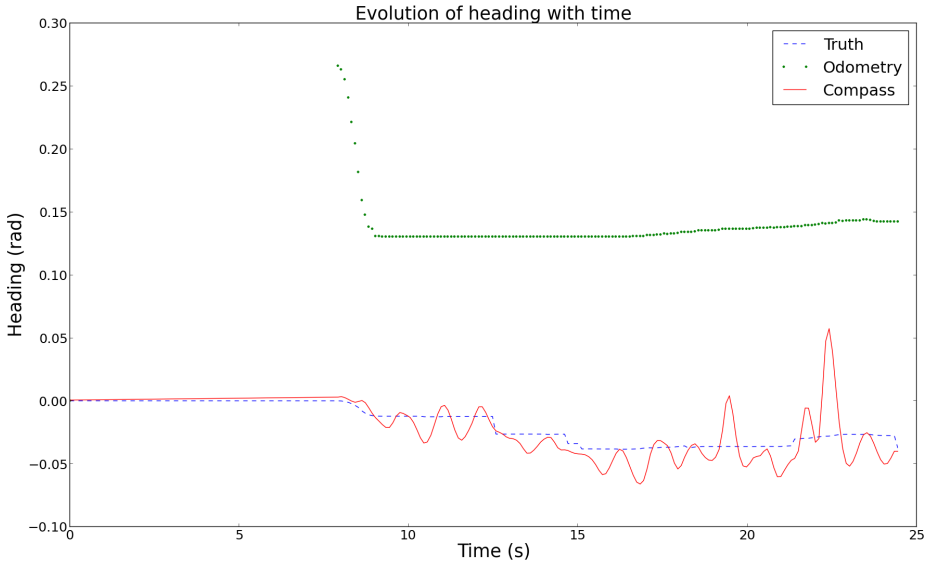


Figure 15: Comparison of robot angle estimations during a translation (25s experiment)

The second experiment makes the robot walk in a straight line for about 25 seconds. Figure 15 shows a comparison of the estimations of the robot base angle (with the same display conventions as the previous figure). This shows that the odometry has a nearly constant bias of approximatively 0.15 radians (around 9°). This also shows that in the ground truth, the robot walks in a near straight line (or at least, a set of parallel lines with a slight lateral shift). The estimated compass angle oscillates around the ground truth, with a weak oscillation amplitude (0.05 radians).

Figure 16 compares the robot trajectory as estimated by the odometry and the ground truth trajectory. The odometry trajectory is drawn as a full blue line, while the ground truth is represented as a dashed green line. The oscillations come from the fact that the robot torso is oscillating during the walk, and the oscillation period corresponds to the stepping period. Note that the odometry underestimates the amplitude of these oscillations. The bias highlighted by Figure 15 can be seen quite clearly, because the trajectory is clearly steadily drifting from the real trajectory, with a constant angle of around 0.12 radians (about 7 degrees).

3.3 Results on the real robot

The compass has been integrated on the robot, with a C++ implementation based on the OpenCV library. Figure 17 shows a typical output of the keypoint matching with the RANSAC model. The

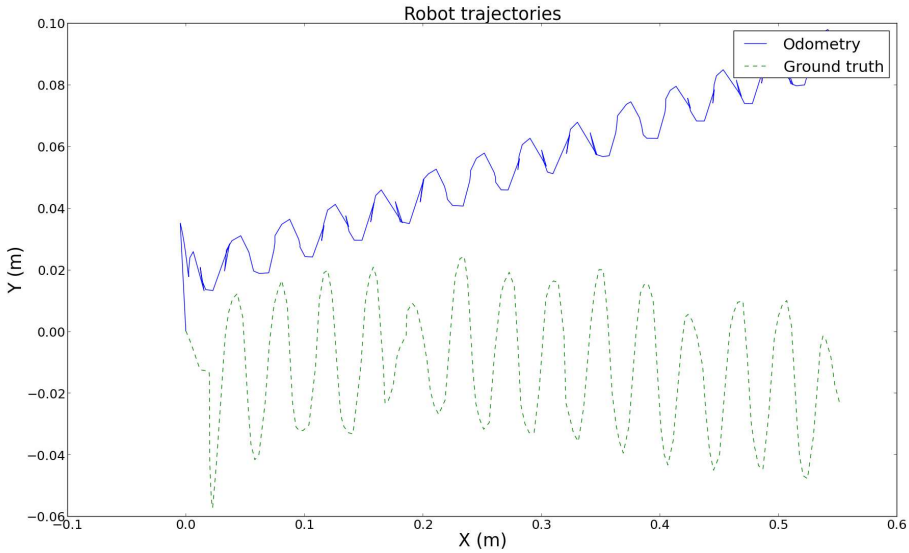


Figure 16: Comparison of robot trajectories estimation during a translation (25s experiment)

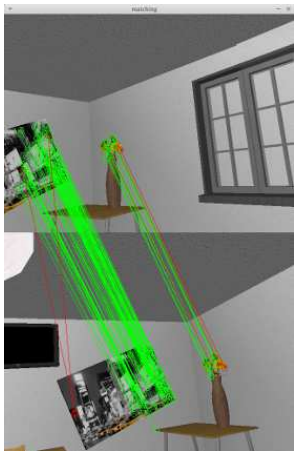


Figure 17: Example of RANSAC output on keypoint matching

reference image is shown on top, and the current one on the bottom part. The inliers are marked in green, while the outliers are drawn in red. This figure shows how all inliers behave consistently, here with a global shift to the left. On the other hand, outliers correspond to inconsistent points, which are due to false matches. Note that the rotation change between the two images is clearly visible in the selected inliers.

Table 1 shows the CPU usage of the algorithm on the robot for available resolutions of the camera. For the walk corrections, the typical parameters is 160x120 and 15fps. The maximum resolution is only available at 5 frames per second, so around 36% CPU. Note that the results are significantly better than in Wirbel et al. (2013), because the keypoints descriptor has been changed from USURF to ORB.

Resolution	160x120	320x240	640x480	1080x960
5 fps	2%	4%	10%	36%
15 fps	6%	11%	31%	NA
30 fps	13%	23%	60%	NA

Table 1: Comparison of CPU usage for different resolution and frame rates

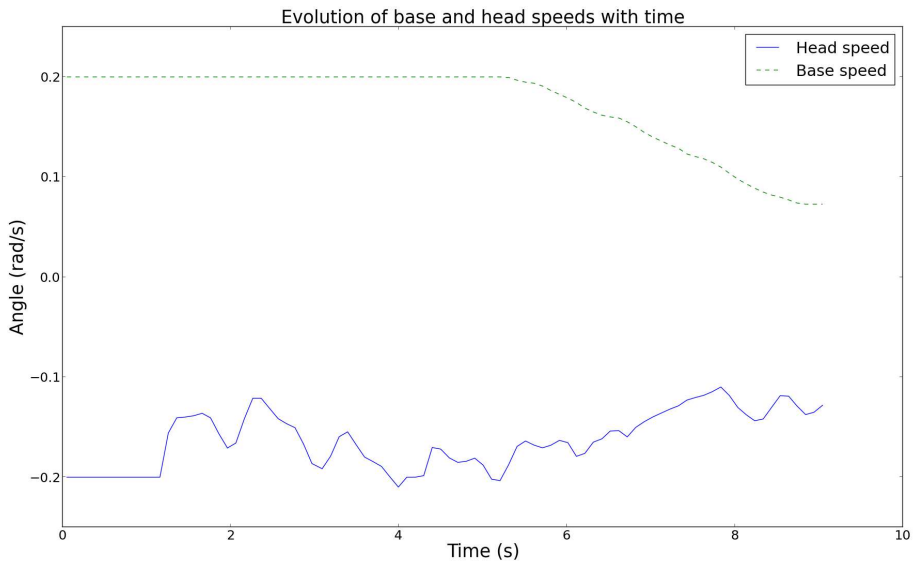


Figure 18: Evolution of head and base speeds with time (9s experiment)

PID parameters have been tuned for the robot. The following results are presented for a desired rotation of $\frac{\pi}{2}$, which lasted around 10 seconds. Figure 18 presents the computed speeds in the robot reference. Both speeds are capped for walk stability. The head speed (blue full line) is drawn is compensating both the base speed and the errors on the head and the base position. The base speed (in green dashed line) is first capped then decreases slowly. When the body angle reaches the destination, the robot is stopped immediately, which explains why the speed is not necessarily zero at the end of the experiments.

Figure 19 shows the evolution of the head angle in the fixed reference. The head angle (in blue full line) is oscillating around the consign (in green dashed line).

Figure 20 shows the evolution of the base angle in the fixed reference. The goal value is $\frac{\pi}{4}$ and the start value is $-\frac{\pi}{4}$. The robot stops when the error is under a fixed threshold. Overshoot is compensated by taking one last measurement at the end of the rotation to retrieve the final angle, which makes it possible to correct this overshoot later on, if necessary.

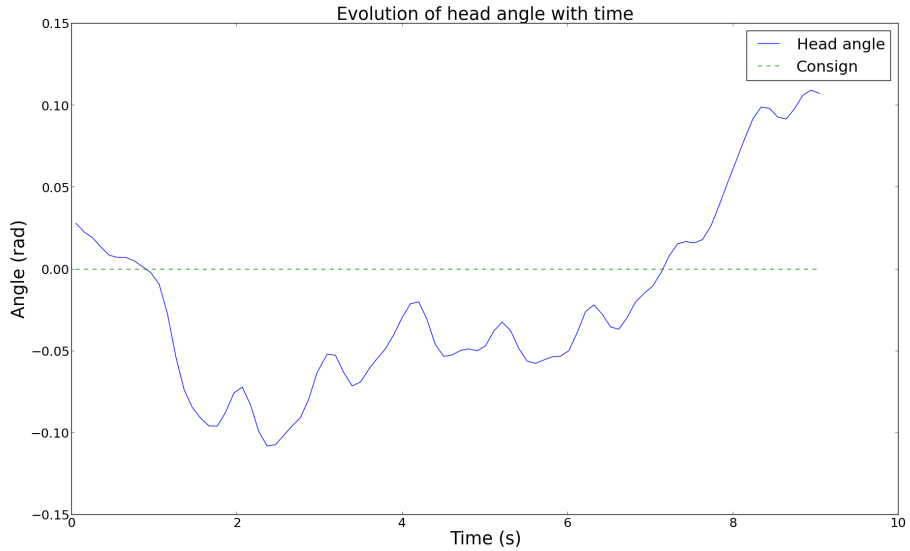


Figure 19: Evolution of head angle with time (9s experiment)

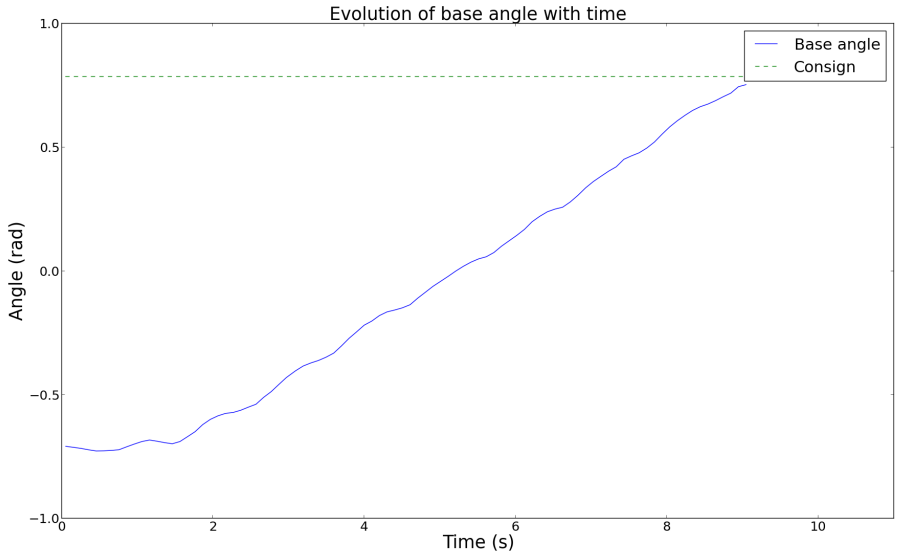


Figure 20: Evolution of base angle with time (9s experiment)

Future plans include to use a motion-capture device to get quantitative and precise data on a real NAO robot, in order to confirm the simulation experiments.

4 Conclusion

In this article, we have presented our work on localization and mapping on a constrained humanoid platform, the NAO robot. We have designed a light and easily interpreted observer to estimate tracked

keypoints positions. We have added a series of adaptations for our platform, including increasing the field of view by moving the head and compensating false matches by checking some consistency conditions and estimation quality. However, the final results have not proven satisfying enough on NAO. This comes in particular from the small robot height, which makes it very difficult to extract high quality keypoints and track them. The robot limited field of view is also an issue, along with its reduced speed, which makes frequent stops long and tedious. It might be possible to reuse some of this work on taller robots, such as ROMEO for example, or some other platform, because there are very few requirements except the ability to track keypoints.

Nevertheless, from our work on keypoint tracking, we have been able to derive a partial localization information, the robot orientation. To do so, we have built an efficient visual compass which is based on matching reference image keypoints with the current observed keypoints, and deducing the current robot rotation. FAST keypoints with ORB descriptors are extracted, and their global rotation is computed using a RANSAC scheme. PID controllers make it possible to control the robot orientation precisely, performing rotations and walking along straight lines with much higher precision that the open loop odometry. The implementation is running in real time on the robot, and its accuracy has been controlled quantitatively on simulator and qualitatively on a robot. Future work includes getting quantitative data on a real robot with motion capture.

In future work we will look more closely at other type of vision based approaches, such as dense visual method base on image correlation, whose characteristics and weaknesses are complementary to the sparse algorithms described in this article.

References

H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer Vision–ECCV 2006*, page 404–417, 2006.

Chun-Hua Chang, Shao-Chen Wang, and Chieh-Chih Wang. Vision-based cooperative simultaneous localization and tracking. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5191 –5197, May 2011. doi: 10.1109/ICRA.2011.5980505.

Denis Chekhlov, Mark Pupilli, Walterio Mayol-Cuevas, and Andrew Calway. Real-time and robust monocular SLAM using predictive multi-resolution descriptors. In *Advances in visual computing*, page 276–285. Springer, 2006. URL `http://link.springer.com/chapter/10.1007/11919629_29`.

Cyberbotics. Webots simulator. URL `http://www.cyberbotics.com/overview`.

A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

D. Gouaillier, C. Collette, and C. Kilner. Omni-directional closed-loop walk for NAO. In *2010 10th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 448 –454, December 2010. doi: 10.1109/ICHR.2010.5686291.

A. Hornung, K.M. Wurm, and M. Bennewitz. Humanoid robot localization in complex indoor environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1690 –1695, October 2010. doi: 10.1109/IROS.2010.5649751.

Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

Georg Klein and David Murray. Compositing for small cameras. In *Proc. Seventh IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'08)*, Cambridge, September 2008.

D. Maier and M. Bennewitz. Real-time navigation in 3D environments based on depth camera data. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)*, 2012.

D. Maier, M. Bennewitz, and C. Stachniss. Self-supervised obstacle detection for humanoid navigation using monocular vision and sparse laser data. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1263 –1269, May 2011. doi: 10.1109/ICRA.2011.5979661.

Y. Matsumoto, K. Sakai, M. Inaba, and H. Inoue. View-based approach to robot navigation. In *2000 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings*, volume 3, pages 1702–1708 vol.3, 2000. doi: 10.1109/IROS.2000.895217.

M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: a factored solution to the simultaneous localization and mapping problem. In *Proceedings of the National conference on Artificial Intelligence*, page 593–598, 2002.

OpenCV. Open computer vision library. URL `http://opencv.org/`.

S. Osswald, A. Hornung, and M. Bennewitz. Learning reliable and efficient navigation with a humanoid. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2375–2380, 2010. doi: 10.1109/ROBOT.2010.5509420.

Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. *Computer Vision–ECCV 2006*, page 430–443, 2006.

Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: an efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, page 2564–2571, 2011.

Sebastian Thrun. A probabilistic on-line mapping algorithm for teams of mobile robots. *The International Journal of Robotics Research*, 20(5):335–363, January 2001. ISSN 0278-3649, 1741-3176. doi: 10.1177/02783640122067435. URL `http://ijr.sagepub.com/content/20/5/335`.

Sebastian Thrun. Robotic mapping: A survey. In *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

E. Wirbel, B. Steux, S. Bonnabel, and A. de La Fortelle. Humanoid robot navigation: From a visual SLAM to a visual compass. In *2013 10th IEEE International Conference on Networking, Sensing and Control (ICNSC)*, pages 678–683, 2013. doi: 10.1109/ICNSC.2013.6548820.