Visual Registering of Arm Pose: a Space Robotics Application

ALDO CUMANI[1], SANDRA DENASI[1], ANTONIO GUIDUCCI[1],
PIERGIORGIO LANZA[2], GIORGIO QUAGLIA[1]
[1]Istituto Nazionale di Ricerca Metrologica
Turin, ITALY
[2]Alcatel Alenia Space Italia
Turin, ITALY

Abstract: This paper deals with the vision algorithms designed for ESA's Eurobot Flight Model, a robotic system planned to assist astronauts on the International Space Station during extra-vehicular activities. The algorithm for registering the robot's pose with respect to the station is presented, together with some preliminary results from laboratory simulations.

Key-Words: Visual registering, Space robotics, Stereo vision

1 Introduction

The ability of a robotic vision sensor to register its relative pose against some known environment is a useful feature for many applications, e.g. for visual servoing in automated manufacturing systems. Such an ability would be useful for space robotics applications too. For example, the International Space Station (ISS) requires Extravehicular Activity (EVA, i.e. spacewalk) in order to support several tasks, like assembly, maintenance, and repair of the station itself and of ISS payloads. An autonomous robotic system, complementary in capability to an EVA crew member, would benefit all these aspects of extravehicular operation.

Such a robot needs visual pose registration for at least two of its intended tasks:

- navigation on the ISS surface, and
- manipulation of items to be serviced.

Indeed, due to their complexity, neither task could be reliably performed on the basis of pre-programmed joint motions alone. On the other hand, periodic visual feedback would allow the robot to proceed autonomously, without need for continuous monitoring by a human operator.

In this context, a robotic system, comparable in size and capability to a crew member, including anthropomorphic arms, has been planned by ESA and is known as the Eurobot Flight Model (EFM). A working Eurobot demonstrator, the Eurobot Wet Model (EWM), is being built in order to have an operational test bed available throughout the Eurobot programme for early operational feedback. The demonstrator shall perform simulated EVA tasks in a pool while controlled by a remote operator, similarly to a crew member in the ISS using an Intra-Vehicular Activity (IVA) workstation. It shall be equipped with a vision system, developed by Alcatel Alenia Space Italia on the basis of a study of the INRIM Computer Vision Group, to perform tasks as robot pose registering, object pose recognition and obstacle detection.

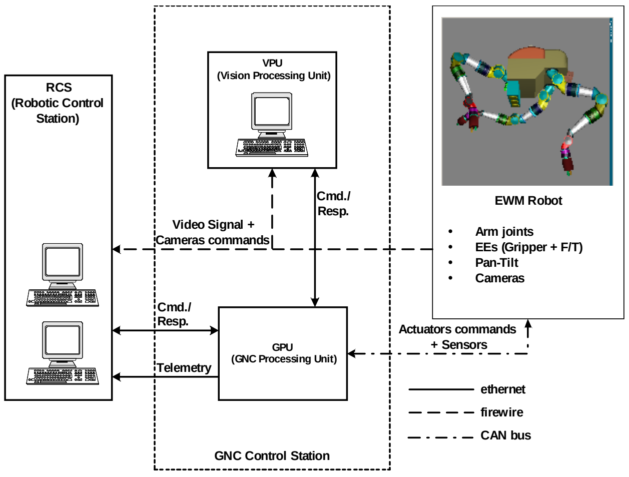


Figure 1: Functional Architecture of the EWM. The main communication lines between the RCS the VPU, GPU and the WET robot are shown in this figure.

The EWM vision system is described elsewhere [1], and its overall architecture is shown in Fig. 1 for the sake of reference.

The EWM has a total of six cameras, three of which mounted on the head through a pan-tilt unit and devoted to navigation and object localization, and the other three on the three robot arms, used for finer localization of serviceable objects and grasping aids. This paper describes in some detail the algorithms used for registering the robot's pose with respect to the station, together with some preliminary results from laboratory simulations.

2 Visual registering of the EWM pose

The problem of registering, from visual data, the relative pose of the vision sensor against a CAD model of the environment has been extensively studied, especially in the field of visual servoing (see e.g. [2, 3, 4, 5, 6]). All published approaches, anyway, are variants of a same basic algorithm which consists in rendering the model from the assumed Point Of View (POV), comparing the result against actual image features and adjusting the POV for the best match.

The registering algorithm proposed for the EWM is no exception, and can be summarized as follows:

- R1. Extraction of features (edge contours) from the actual image.
- R2. Wireframe rendering, with hidden line removal, of the CAD model of the environment as seen from the current estimated POV.
- R3. Matching of observed image edges to rendered model edges and computation of an appropriate error function measuring the mismatch.
- R4. Update of the POV estimate by minimisation of the above error function.
- R5. Loop from R2 (or R3) until convergence or for a predefined number of iterations.

As this algorithm requires the use of a calibrated sensor, to the above one should logically add a preliminary step

- R0. Camera calibration.

which, however, needs only be performed once per robot mission.

In the following, the above steps are described in some detail.

2.1 Calibration

The calibration phase actually consists of two different tasks, namely

a) intrinsic geometric calibration of each camera or camera subsystem, and
b) hand-eye calibration.

Intrinsic calibration is achieved by means of a state-of-the art IAC-based algorithm like Bouguet's [7] or Zhang's [8], making use of one or more chessboard patterns fixed on the ISS and taking advantage of the mobility of the cameras. Hand-eye calibration is performed by the classic Tsai-Lenz method [9], which again needs only a chessboard pattern observed from several poses.

2.2 Pre-Processing.

For each acquired image, step R1 actually consists of two subsequent steps, namely 1) removal of lens distortion and 2) extraction of useful image features.

Distortion removal consists in warping the input image by applying the inverse distortion model, as estimated during the calibration step. A convenient implementation of such warping process is via bilinear interpolation on a lookup table. The output of this step is an image as would be obtained by a camera obeying the pure pinhole model.

Feature extraction consists in locating those pixels or aggregates of pixels in an image that have some distinctive characteristics. In a man-made environment like the ISS, the most useful features are apparent object edges, i.e. image contours and segments of such contours.

The algorithm used for contour extraction is based on the well established paradigm that defines contours as lines of maximal intensity variation across the line itself. It works by first smoothing the intensity image by a Gaussian filter, and then finding zero crossings of the second directional derivative of the smoothed luminance along the luminance gradient direction [10, 11].

Contour lines can then be segmented into straight or curvilinear segments (see Fig. 2). The minimal information needed to describe each segment (e.g. endpoint coordinates for a straight segment) can be augmented by attributes describing the luminance transition across the segment (like mean luminance, transition amplitude and sharpness), possibly useful for a more reliable match against the CAD model. The current implementation, however, only uses straight segments and does not include photometric attributes, since due to uncertainty in the actual illumination, the latter would not be of much use.

Finally, feature coordinates are transformed, using the intrinsic camera parameters, into calibrated image coordinates (i.e. corresponding to unit focal length).

2.3 Rendering

As concerns step R2, rendering with hidden line removal relies on the use of an efficient representation of

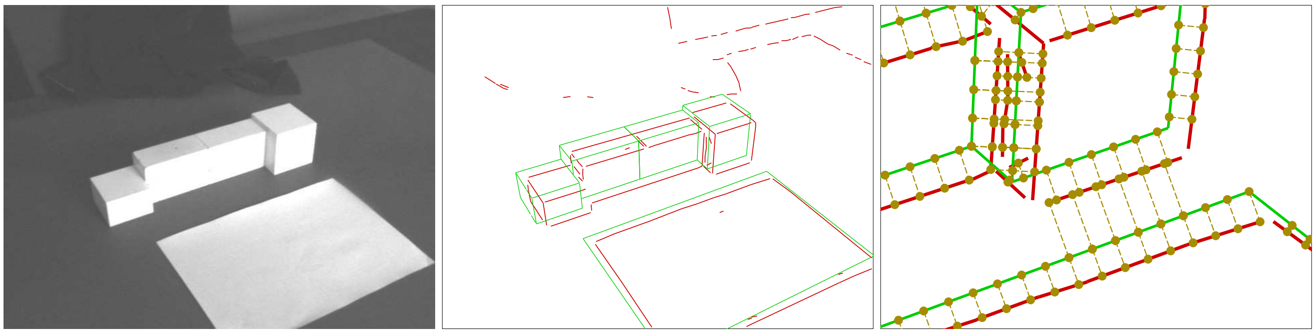


Figure 2: Left: example image. Middle: image segments (dark red) and rendered model edges (light green). Right: detail showing matched points.

the CAD model such as the one provided by a Binary Space Partition Tree [12]. Transforming a given CAD model into a BSPT is computationally expensive, but since the ISS environment is almost unchanging such computations may be done beforehand.

Given the BSPT, its traversal automatically yields the model primitives in the correct order (front to back) for 2D rendering. Hidden line removal is achieved by depth rendering of each BSPT triangle onto a z-buffer of suitable size; each new BSPT edge to be rendered is sampled against the depth map stored in the z-buffer to determine its visible parts. The algorithm must also properly take into account model edges that are only visible if they concide with occluding borders, as e.g. those of the triangular facets used to approximate a curved surface.

Note that in our case rendering differs from the standard way in that the 3rd (depth) component of each visible model edge is kept; this allows to compute the derivatives, needed by the optimization step (R4), of 2D edge positions with respect to the POV parameters.

2.4 Matching

Matching is accomplished by taking several equispaced points on each projected model edge, and searching for detected image edges (segments) orthogonally to the model edge and within a specified distance (see Fig. 2, right). This is similar to [6], although those authors do not use precomputed edge features but instead perform a search for intensity variations directly on the image.

2.5 Optimisation

Finally, step R4 defines a cost function as the sum of the (suitably weighted) squared 2D distances of matched edge points, linearizes this cost with respect to the six POV parameters (three for rotation and three for translation), and finds a correction to the current POV by solving the latter Linear Least-Squares problem. The linear parametrization of the cost function is made possible by the fact that the rendered edges are known in full 3D, as said before. Indeed, if $\mathbf{X} = [x, y, z, t]^\top$ are the 3D (projective) CAD coordinates of some world point (e.g. one of the sampled model points) and $\mathbf{X}_C = [x_C, y_C, z_C, t_C]^\top$ its coordinates in the camera reference,

$$\mathbf{X}_C = \mathsf{M}_\delta \mathsf{M}_{cur}(\mathsf{M}_0 \mathbf{X}) \tag{1}$$

where M_0 is the known nominal world-camera transformation, derived from the assumed robot pose, M_{cur} the current correction (from the previous iteration, $\mathsf{M}_{cur} \equiv \mathsf{I}$ at start) and M_δ the sought-for optimizing correction. All these are 4×4 Euclidean transformation matrices of the form

$$\mathsf{M} = \begin{bmatrix} \mathsf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} = \begin{bmatrix} e^{[\mathbf{r}]_\times} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \tag{2}$$

where $\mathbf{r}$ is the vector representation of rotation R, i.e. $\mathbf{r}/\|\mathbf{r}\|$ = rotation axis, $\|\mathbf{r}\|$ = rotation angle, and $[\mathbf{r}]_\times$ is the antisymmetric matrix representation of vector cross product by $\mathbf{r}$. Note that the term $(\mathsf{M}_0\mathbf{X})$ in Eq. (1) corresponds to the view from the nominal assumed robot pose, i.e. to the actual output of the rendering procedure.

Therefore, if $\mathbf{x}_m = [x_C/z_C, y_C/z_C]^\top$ are the (calibrated) image coordinates of a projected model point (from Eq. (1)) and $\mathbf{x}_s$ those of the matched contour point, we can define a fitting criterion

$$J(\mathbf{p}_\delta) = \sum_i f(\|\mathbf{x}_{mi} - \mathbf{x}_{si}\|^2) \tag{3}$$

which is parametrized in terms of the six unknowns $\mathbf{p}_\delta = [\mathbf{r}_\delta^\top, \mathbf{t}_\delta^\top]^\top$ which generate the correction transform M_δ and is easily linearized with respect to $\mathbf{p}_\delta$. Note that $f(\cdot)$ = identity corresponds to standard linear least squares; for reasons of robustness against outliers (mismatched points), it would be preferable to use a robust cost function such as the Lorentzian cost:

$$f(e^2) = \log(1 + e^2/\sigma^2) \qquad (4)$$

In our implementation we simulate a Lorentzian cost function by using suitable error weights, i.e.

$$f(e_i^2) = e_i^2/(1 + e_{ip}^2/\sigma^2)$$

with e_{ip} the error values from the previous iteration.

2.6 Iteration

In the current implementation, iteration looping restarts from step R3, after incorporating the last computed M_δ into M_{cur}. Indeed, assuming small POV errors, the visual structure of the rendered model edges should be still valid for the corrected POV (i.e. no big changes in edge visibility). If this is not the case, as indicated e.g. by large error residuals, the iteration may restart from step R2, with a new rendering of the model from the corrected POV (i.e. after incorporating M_{cur} into M_0).

2.7 Multiple cameras

In the above, the algorithm has been exposed with reference to a single camera. However, the same approach is readily extended to a system with any number of rigidly linked cameras, provided such system is calibrated (i.e. the relative poses of the cameras must be known). For a system with N cameras C_k, $k = 0 \dots N-1$, Eq. (1) becomes

$$\begin{aligned} \mathbf{X}_{C_k} &= \mathsf{M}_{C_k}\mathsf{M}_\delta \mathsf{M}_{cur}\mathsf{M}_0\mathbf{X} \\ &= \mathsf{M}_{C_k}\mathsf{M}_\delta \mathsf{M}_{cur}\mathsf{M}_{C_k}^{-1}(\mathsf{M}_{C_k}\mathsf{M}_0\mathbf{X}) \end{aligned} \qquad (5)$$

where M_{C_k} represents the pose of camera C_k relative to the camera system, and the term $(\mathsf{M}_{C_k}\mathsf{M}_0\mathbf{X})$ corresponds to the rendered model as seen from the nominal pose of C_k. Substituting Eq. (5) instead of Eq. (1) into the definition of the cost $J(\mathbf{p}_\delta)$ and summing up the contributions of all cameras yields an optimization problem with the same structure as in Sec. 2.5 and that can be solved analogously.

It should be remarked that, unlike a stereo approach, this one does not require view-to-view matching, so it is not necessary that all objects be visible in all views (in fact, each camera could frame a completely different portion of the environment).

3 Laboratory tests

Some preliminary tests on the visual registration algorithm have been carried on at INRIM. The sensor was a CCD camera, providing VGA-resolution images (640×480 pixels), mounted on the end effector of a Samsung AW1 lightweight arm, and framing a scene with a typical ISS handrail as shown in Fig. 3.

Camera calibration was done by Bouguet's algorithm, using a chessboard pattern printed on a standard A4 sheet of paper, grabbed from five different poses. Hand-eye calibration was done by Tsai's algorithm, again using the same pattern as target.

The test reported here consisted in grabbing the scene from several (40) randomly scattered positions, as shown in Fig. 4. In the latter, the asterisk marks the assumed origin of the World Coordinate System (WCS), located near the center of the handrail, and nominally at (1.42,-0.04,0.515) in the Robot Coordinate System (RCS). The WCS and RCS axes are nominally assumed parallel. Note that the "outliers" mentioned in the figure are two points which were discarded because the arm was not able to reach the planned pose due to joint constraints.

The results of running the registration algorithm for each of these positions are shown in Figs. 5 and 6, which report the positional and angular correction to the assumed relative pose of robot and world (handrail), that is the transformation between the Robot Coordinate System (RCS) and the World Coordinate System (WCS). These results show that this approach is able to determine the robot position with an uncertainty of the order of 2-3 mm, and its orientation within some 0.1 deg.

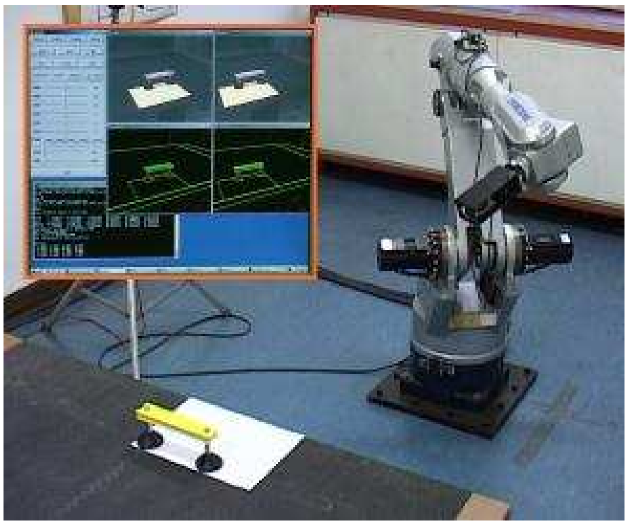


Figure 3: Hardware setup and man-machine interface for pose registration tests with a camera on the robot wrist.

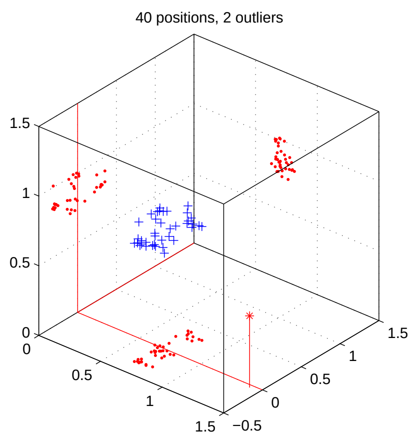
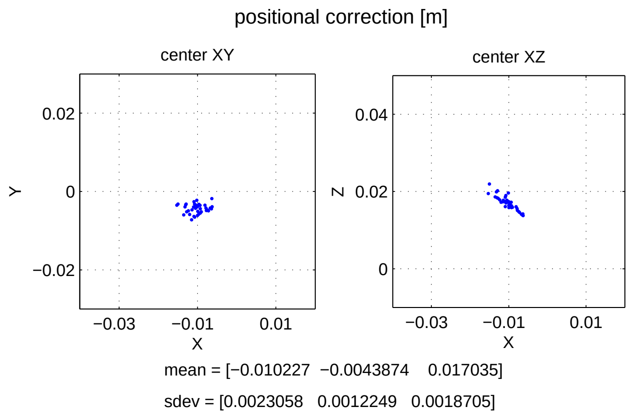


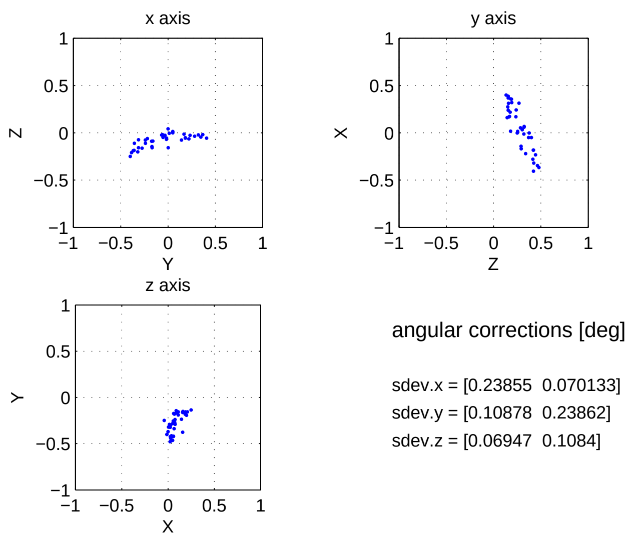
Figure 4: Wrist positions in RCS (blue crosses) and their projections on the coordinate planes (red dots). The asterisk marks the assumed origin of WCS (near the center of the handrail).



mean = [−0.010227 −0.0043874 0.017035]

sdev = [0.0023058 0.0012249 0.0018705]

Figure 5: Estimated positional correction to the origin of WCS.



sdev.x = [0.23855 0.070133]

sdev.y = [0.10878 0.23862]

sdev.z = [0.06947 0.1084]

Figure 6: Estimated angular correction to WCS orientation. Each plot represents the components of a WCS axis on the plane orthogonal to the corresponding RCS axis.

References:

[1] P. G. Lanza, A. Cumani, S. Denasi, A. Guiducci, and G. Quaglia, “Image processing applied on the WET robot prototype,” in *Proc. Intl. Space System Engineering Conf. DASIA 2006*, 2006.

[2] D. G. Lowe, “Fitting parameterized three-dimensional models to images,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 441–450, May 1991.

[3] M. Armstrong and A. Zisserman, “Robust object tracking,” in *Proceedings of the Asian Conference on Computer Vision*, pp. 58–61, 1995.

[4] G. D. Hager, S. Hutchinson, and P. Corke, “A tutorial on visual servo control,” *IEEE Trans. Robot. Automat.*, vol. 12, no. 5, pp. 651–670, 1996.

[5] F. Martin and R. Horaud, “Multiple-camera tracking of rigid objects,” Tech. Rep. RR-4268, INRIA, 2001.

[6] T. Drummond and R. Cipolla, “Real-time visual tracking of complex structures,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, 2002.

[7] J. Y. Bouguet, “Complete camera calibration toolbox for MATLAB,” tech. rep., `http://www.vision.caltech.edu/bouguetj/calib_doc/index.html`, 2004.

[8] Z. Zhang, “A flexible new technique for camera calibration,” Tech. Rep. MSR-TR-98-71, Microsoft Research, 1998.

[9] R. Tsai and R. Lenz, “A new technique for fully autonomous and efficient 3d robotics hand/eye calibration,” *IEEE Trans. Robotics Automat.*, vol. 5, no. 3, pp. 345–358, 1989.

[10] P. Grattoni and A. Guiducci, “Contour coding for image description,” *Pattern Recognition Letters*, vol. 11, no. 2, pp. 95–105, 1990.

[11] A. Cumani, “Edge detection in multispectral images,” *CVGIP: Graphical Models and Image Processing*, vol. 53, no. 1, pp. 40–51, 1991.

[12] M. Paterson and F. Yao, “Efficient binary space partitions for hidden surface removal and solid modeling,” *Discrete and Computational Geometry*, vol. 5, pp. 485–503, 1990.