Robotica (2005) volume 23, pp. 365–375. © 2005 Cambridge University Press
doi:10.1017/S0263574704001006 Printed in the United Kingdom

Real-time visual tracking based on BSP-tree representations of object boundary

Vincenzo Lippiello

PRISMA Lab, Dipartimento di Informatica e Sistemistica, Università degli Studi di Napoli Federico II, Via Claudio 21, 80125 Napoli (Italy) E-mail: vincenzo.lippiello@unina.it

(Received in Final Form: July 29, 2004)

SUMMARY

In this paper an algorithm for real-time estimation of the position and orientation of a moving object using a video camera is presented. The algorithm is based on the extended Kalman filter which iteratively computes the object pose from the position measured in the image plane of a set of feature points of the object. A new technique is proposed for the selection of the optimal feature points based on the representation of the object geometry by means of a Binary Space Partitioning (BSP) tree. At each sample time, a visit algorithm of the tree allows pre-selecting all the feature points of the object that are visible from the camera in the pose predicted by the Kalman filter. A further selection is performed to find the optimal set of visible points to be used for image feature extraction. Experimental results are presented which confirm the feasibility and effectiveness of the proposed technique.

KEYWORDS: Visual tracking; Object boundary; Real-time estimation; BSP tree; Kalman filter.

1. INTRODUCTION

The use of visual sensors may have a high impact in applications where it is required to measure the pose (position and orientation) and the visual features of objects moving in unstructured environments. Typical industrial applications are assembling of mechanical parts, edge following, object grasping; non industrial applications are automotive guidance, spatial and underwater robotics. In fact, visual sensors offer the possibility to extract multiple information from a workspace in a noninvasive manner. Moreover, in the last decade, high performance vision systems are becoming less and less expensive. This scenario is quite appealing for industries and researchers which are stimulated at developing innovative strategies that actual technology is able to support.

In robotics, the measurements provided by video cameras can be directly used to perform closed-loop position control of the robot end effector, usually denoted as visual servoing control.[1] In this case, the position tracking error can be defined either in the Cartesian space (position-based visual-servoing[2]) or in the image space of the cameras (image based visual servoing[3,4]). More recently, vision measurements have been used in combination with force measurements to develop control strategies aimed at improving the robot performance for the execution of tasks in scarcely structured environments.[5]

A strict requirement in visual servoing applications is that the information extracted from visual measurements must be available at control sampling rate. In particular, for position-based visual servoing, computational efficient visual tracking techniques, able to estimate the position and orientation of moving objects, must be adopted. Moreover, visual measurements are usually affected by significant noise and disturbances due to temporal and spatial sampling and quantization of the image signal, lens distortion, etc., which may produce large errors in the pose estimation. The adoption of the extended Kalman filter represents a good trade-off between computational load and estimation accuracy.[6–9]

In fact, Kalman filtering offers many advantages over other pose estimation methods,[10–12] e.g. implicit solution of photogrammetric equations with iterative implementation, temporal filtering, ability to change the measurement set during the operation. Moreover, the statistical properties of Kalman filter may be tuned to those of the image measurements noise of the particular vision system. Last but not least, the prediction capability of the filter allows setting up a dynamic windowing technique of the image plane which may sensibly reduce image processing time. Applications of a Kalman filter in machine vision range from visual tracking of objects with many internal degrees of freedom,[13] to automatic grasp planning[14] as well as pose and size parameters estimation of objects with partially known geometry.[15]

A widely adopted strategy that can be used to estimate the object pose is based on the recognition of some geometric features of the object, such edges and corners, from a camera image. In particular, strategies based on the extraction of a suitable number of corners (feature points) allow the object pose to be computed using a simple point CAD model.[2,14] In principle, the accuracy of the estimate increases with the number of the available feature points, at the expense of the computation time. However, when Kalman filter is used, it has been shown that the best achievable accuracy can be obtained using a number of five or six feature points, if properly chosen.[7]

Suitable selection algorithms have been developed to find the optimal feature points for pose estimation.[16,17] It should be pointed out, however, that the computational complexity of these algorithm grows at factorial rate and thus, in case of objects with a large number of corners, it is crucial to perform a pre-selection, e.g. by eliminating all the points that are occluded with respect to the camera.[18,19]

In this paper, a new pre-selection technique of the feature points is proposed, based on the detection at each sample time of all the points that are to be visible to the camera at next sample time. This algorithm exhibits a complexity which grows linearly, thanks to the use of Binary Space Partitioning (BSP) tree to represent the object geometry.[20] Differently from reference [20], the BSP Tree data structures are used to represent the object geometry using only the object corners which may be also used for the pose reconstruction process. In this way, the computational complexity of the proposed pre-selection algorithm is very small because it depends on operations that work on geometrical zero-dimensional data. In detail, the prediction of the object pose provided by the Kalman filter is used to drive a visit algorithm of the BSP tree which allows identifying all the feature points that are to be visible at the next sample time. After the pre-selection, an optimal point selection algorithm and a dynamic windowing algorithm are adopted to find the windows of the image plane to be considered for feature extraction.

The proposed pre-selection technique can be used also in the case of objects and obstacles with interposing parts.[21] It should be pointed out that only few real-time implementations of algorithms for visibility determination have been proposed so far. Some of them consider occlusion only as a simple case of a single convex object,[22] others are capable to track the 3D object pose only for object moving at a fixed distance from the camera.[23] A more efficient model-based algorithm for dynamic handling of occlusions was developed in reference [24], where the prediction of the object pose provided by a Kalman filter is used for hidden parts removal based on a partial 3D reconstruction of the image features (lines and ellipses). More recently, a visual tracking algorithm where BSP trees are used for real-time removal of hidden lines have been proposed.[25]

The present work combines the computational efficiency of the operations on Binary Space Partition trees with the simplification introduced by the use of point features (object corners), which are easy to identify using small windows and can be extracted with high robustness in various environmental conditions.[17] The effectiveness of the proposed approach is tested in experimental case studies where the position and orientation of an object carried by a robot manipulator is estimated using one fixed camera.

The paper is organized as follows. In Section 2 the BSP tree representation of the geometry of an object is derived using a suitable object boundary representation; in Section 3 the proposed pre-selection algorithm is illustrated; the selection algorithm is described in Section 4, and the whole estimation procedure is summarized in Section 5; the experimental results are presented in Section 6, while in Section 7 some concluding remarks and future perspectives are reported.

2. MODELING

The well known pin-hole model is considered to model a camera fixed with respect to the base coordinate frame. The details of the adopted camera model are presented in detail in reference [26].

The pose reconstruction process is based on a suitable Kalman filter formulation that considers as input the image projections of the object corners and provides as output the corresponding object pose. Since the output model of the adopted Kalman filter formulation is nonlinear in the system state, it is required to linearize the output equations about the current state estimate at each sample time, considering the extended Kalman filter (see reference [9] for the extended Kalman filter equations).

2.1. 3D object modeling

Modeling the geometry of the objects is a crucial step involved into an object-oriented machine vision process. The realization of a task requiring tracking of target objects requires the knowledge of the geometry of the objects and their relative poses in the observed workspace. If such information has to be provided by a visual system, then a model of the geometry of the target objects, suitably defined for the kind of involved image elaboration process, is required. Unfortunately, in most cases a Cartesian CAD model (*boundary representation* or *B-reps*) of the objects is available (or derivable), but it cannot be directly employed by sophisticated real-time image processing algorithms, while a more efficient representation of the object geometry is necessary to reduce computational complexity. As a matter of fact, two different requirements have to be satisfied: it is important to have an *easily derivable representation* that allows a manual description of the geometry of the object; also, it is necessary to have an *efficient representation* that allows the implementation of real-time visual servoing algorithms.

A good trade-off between these demands may be obtained using a simple boundary representation to describe the object geometry, and a more complex and computationally efficient object representation that may be automatically derived from the previous one. To achieve this goal the so called *manmade* object class is considered. This class contains all objects which may be represented (or approximated) as a union of planar surfaces, e.g. polyhedral objects. Further, the contours of each surface of these objects have to be representable (or approximable) with a polygonal determined by an ordered sequence of points. This means that a curved contour has to be approximated as a polygonal contour via a suitable spatial sampling.

In the following, a possible boundary representation of manmade objects and a specific data structure representing a recursive and hierarchical partition of an n-dimensional space into convex subspaces, known as *BSP tree*, are presented. In particular, the chosen boundary representation is very simple and accessible for a manual geometric description of the object, and the proposed BSP tree data structure may be automatically derived from it.

The considered boundary representation has three specific properties:

https://doi.org/10.1017/S0263574704001006 Published online by Cambridge University Press

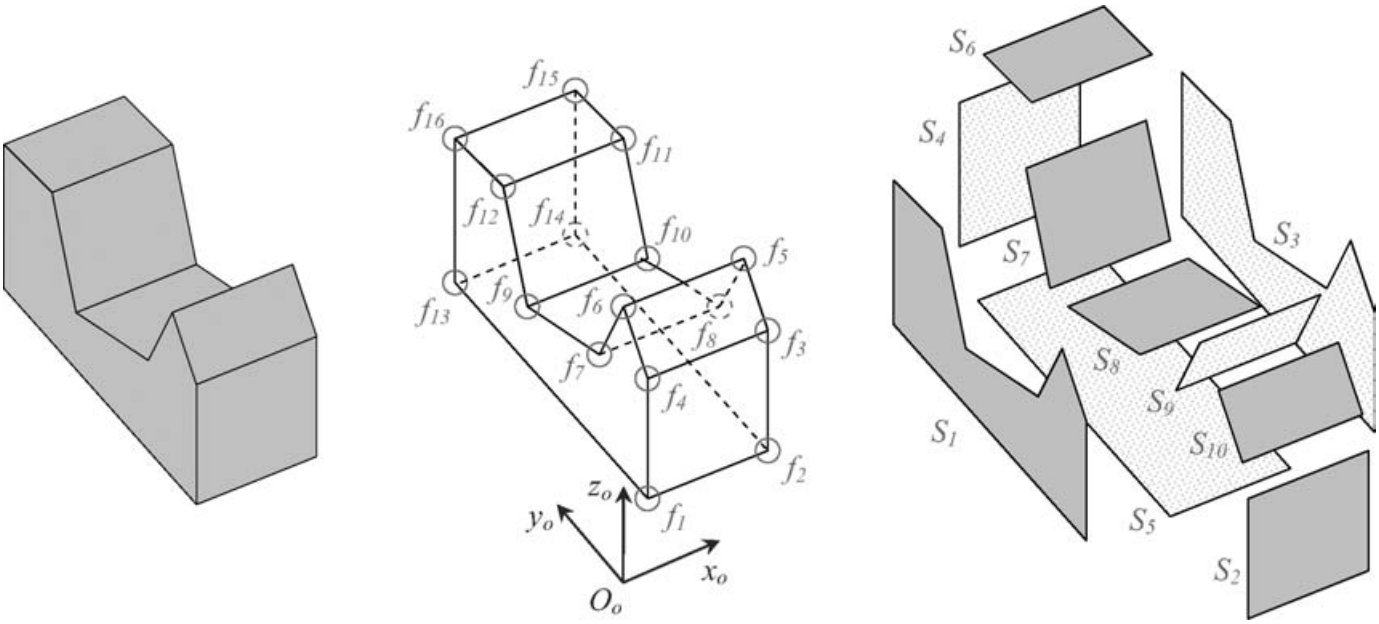

Fig. 1. Example of Cartesian CAD model of a 3D manmade object. Left: object; center: object feature points (First level of representation); right: object surfaces (Second level of representation).

- it is suitable for representing manmade objects;
- it has to be easily constructed manually;
- it has to describe the object with two hierarchical levels of representation: the top level describes the set of the *feature points* of the object, while the bottom level describes the object surfaces using the information of the first level.

In this work, the so-called *feature points* are all the points which allow the contours of the object to be fully described. Generally, they are found by the discontinuities in the direction of the contours and by the intersection of multiple contours. In the case of curved contours, a suitable spatial sampling is required to approximate the contour with a polygonal line.

The considered B-reps may be constructed in two steps. During the first step all the feature points of the object are individuated and measured with respect to a chosen object frame, fixed with the object. During the second step an easy description of the object surfaces is achieved by choosing the sequences of feature points delimiting the considered surface. The sequences of feature points are ordered in anticlockwise direction with respect to the outgoing unit vector orthogonal to the object surface. With this choice, it will be possible to recognize the external side of a surface.

In Fig. 1 an example of a manmade object is shown, along with its feature points and its surfaces. In the first level of representation, 16 feature points have been found, which are sufficient to represent all the contours of the object. Each point (f_i, with $i = 1, \ldots, 16$) is represented by its position with respect to the object frame. In the second level of representation 10 surfaces are found and described using the previous feature points. Table I reports the representation of the surfaces with the corresponding ordered sequences of feature points, as described above.

Notice that it is not important which is the first point of the sequence but only its order.

Table I. Boundary representation. Ordered sequences of feature points corresponding to the surfaces of the object of Fig. 1.

Surface	Feature points
S_1	$\{f_1 f_4 f_6 f_7 f_9 f_{12} f_{16} f_{13}\}$
S_2	$\{f_1 f_2 f_3 f_4\}$
S_3	$\{f_2 f_{14} f_{15} f_{11} f_{10} f_8 f_5 f_3\}$
S_4	$\{f_{16} f_{15} f_{14} f_{13}\}$
S_5	$\{f_1 f_{13} f_{14} f_2\}$
S_6	$\{f_{12} f_{11} f_{15} f_{16}\}$
S_7	$\{f_9 f_{10} f_{11} f_{12}\}$
S_8	$\{f_9 f_7 f_5 f_{10}\}$
S_9	$\{f_6 f_5 f_8 f_7\}$
S_{10}	$\{f_6 f_4 f_3 f_5\}$

2.2. BSP tree geometric modeling

A BSP tree is a data structure representing a recursive and hierarchical partition of a n-dimensional space into convex subspaces[27] which can be effectively adopted to represent the 3D CAD geometry of a set of manmade objects. Each object surface is a polygon, characterized by a set of feature points (the vertices of the polygon) and by the vector normal to the plane leaving from the object. Without loss of generality, the case of a single object is considered.

The BSP-tree structure is a binary tree whose nodes are composed by four elements:

- the partition plane;
- the set of surfaces contained on the partition plane;
- the link to the front sub-tree;
- the link to the back sub-tree.

The partition plane is used to divide the 3-dimensional space into two sub-spaces; it is characterized by a point on the plane and by the unit vector normal to the plane. The sub-space containing the normal vector is named front sub-space,

https://doi.org/10.1017/S0263574704001006 Published online by Cambridge University Press

while the other is named back sub-space. For the purpose of this work, the partition plane has to be chosen in the set of the planes containing the polygons corresponding to the object surfaces; hence, the partition plane contains at least one polygon of the object.

Each node is the root of two subtrees: the front subtree corresponding to a subset of polygons lying entirely in the front sub-space, and the back sub-tree, corresponding to a subset of polygons lying entirely in the back sub-space.

In fact, in order for a BSP tree to represent a solid object, each cell of the tree must be classified as being either entirely inside or outside of the object; thus, each leaf node corresponds to either an *in-cell* or an *out-cell*. The boundary of the set then lies between in-cells and out-cells; since the cells are bounded by the partitioning hyperplanes, it is necessary for all of the boundaries to lie in the partitioning hyperplanes.

Therefore, it can convert from a boundary representation to a tree simply by using all the face hyperplanes as partitioning hyperplanes (*auto-partition*). The face hyperplanes can be chosen in any order and the resulting tree will always generate a convex decomposition of the internal side and the external side. If the hyperplane normals to the boundary representation faces are consistently oriented to point to the external side, then all *back* leaves will be in-cells and all *front* leaves will be out-cells.

However, the choice of partition plane will strongly affect the results. If a polygon happens to span the partition plane, it will be split into two or more pieces. A poor choice of the partition plane can result in many such splits, and a marked increase in the number of polygons. Usually there will be some trade off between a well-balanced tree and a large number of splits.

The problem is that polygons get split during the construction phase, which may give rise to a larger number of polygons. Larger numbers of polygons translate into larger storage requirements and longer tree traversal times. This is undesirable in all applications of BSP Trees, and thus some scheme for minimizing splitting will improve tree performance. If should be considered that minimization of splitting requires pre-existing knowledge about all the polygons that will be inserted into the tree. This knowledge may not be available for interactive use such as solid modelling, but it is available for the application considered in this paper. The easiest strategy to minimize splitting is to choose a partition plane, if it may be extracted from the list of the remaining polygons, that does not intersect with any other polygon. This strategy has a visibility on the recursive steps of only one step, and thus it does not assure that the obtained tree contains the minimal number of polygons. However, some other more sophisticated strategy may be applied to impact this problem.

The BSP tree can be built using a recursive procedure described in the following Pascal-like code:

```
procedure Build_BSP_tree(node:BSP_tree;polygons:
polygon_list); var poly = polygon;
var front_surfaces, back_surfaces = polygon_list;
begin
    {get a surface from the list}
    poly:=get\_polygon(polygons);
    {add poly to the list of surfaces of the current node}
    add_to_list(node->surfaces,poly);
    {compute the partition plane}
    node->partition\_plane:=get_plane(poly);
    {classify remaining polygons with respect to
        the partition plane}
    poly:=get\_polygon(polygons);
    while (poly NOT NULL) do}
        case classify_polygon(poly,node->partition_plane)
            COINCIDENT:
            add_to_list(node->surfaces,poly);
            IN_FRONT_OF:
            add_to_list(front_sufaces,poly);
            IN_BACK_OF:
            add_to_list(back_sufaces,poly);
            INTERSECTING:
            split_polygon(poly,node->partition_plane,
                front_surfaces,back_surfaces);
        end {case}
        poly:=get_polygon(polygons);
    end {while}
    if front_surfaces NOT EMPTY then
        {build front sub-tree}
        node->front\_link:=allocate_node();
        Build_BSP_tree(node->front_link;front_surfaces);
    end {if}
    if back_surfaces NOT EMPTY then
        {build back sub-tree}
        node->back_link:=allocate_node();
        Build_BSP_tree(node->back_link;back_surfaces);
    end {if}
end {begin}
```

In the above procedure, the function `get_polygon()` extracts a polygon from the input list of polygons; the first extracted polygon is used to compute the partition plane (so that it contains the polygon).

The function `get_plane()` computes the partition plane for the current node, i.e., computes the vector $\boldsymbol{p} = [a \;\; b \;\; c \;\; d]^T$ of the coefficients of the equation of the plane containing the input polygon

$$ax + by + cz + d = 0,$$

where $\boldsymbol{n} = [a \;\; b \;\; c]^T$ is the unit vector normal to the input polygon.

The function `classify_polygon()` allows to determine if the current polygon is coincident, in front of, in back of, or intersect the partition plane, in order to complete the list of polygons of the current node and generate the lists of polygons of the front sub-tree and of the back-subtree. When a polygon intersects the partition plane, it is split into two polygons using the procedure `split_polygon()` and the resulting parts are added to the corresponding lists.

Notice that the procedure is recursive and ends when all the polygons and their parts are placed in a node of the tree.

As an example, consider the object represented in Fig. 2, which contains ten polygons. A possible BSP-tree representation of the object is reported in Fig. 3, which has been obtained by considering as root node the partition plane containing polygon number 10. Notice that different choices of the initial node correspond to different trees.

The front sub-tree departing from the root node is empty while the back sub-tree contains all the remaining polygons. The partition plane of the back sub-tree contains the polygon number 1; its the front sub-tree is empty while the back sub-tree contains the polygons from number 2 to number 9.

https://doi.org/10.1017/S0263574704001006 Published online by Cambridge University Press

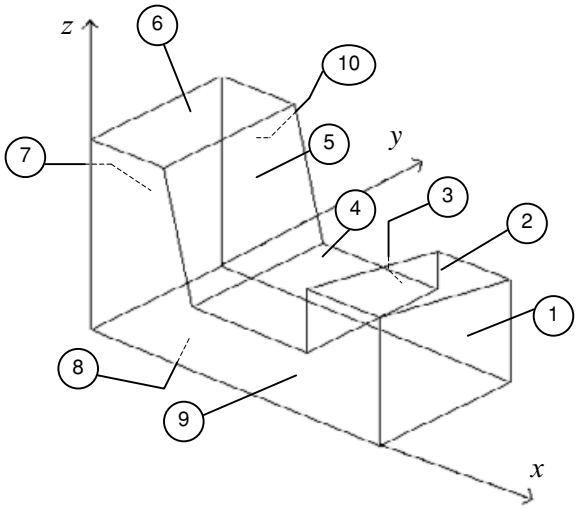

Fig. 2. Object and corresponding polygons.

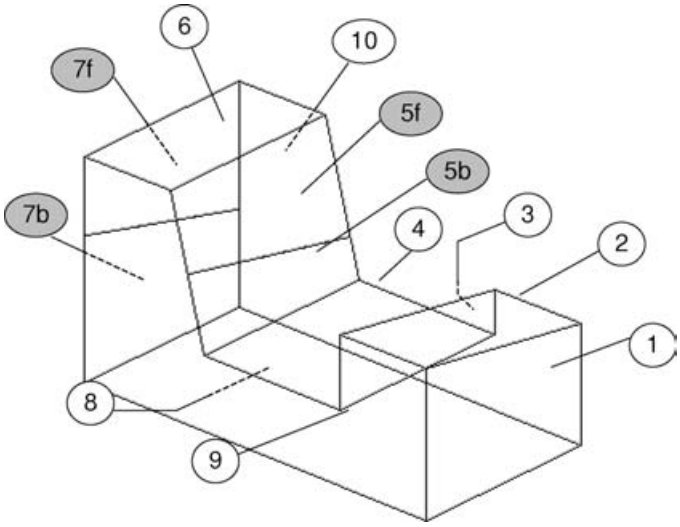

Fig. 4. Case of partition plane splitting polygons number 5 and number 7.

Remarkably, the partition plane containing the polygon number 2 intersects polygons number 5 and 7 (notice that polygons number 9 and 10, which also intersect the partition plane, were already added to previous nodes of the tree and therefore were deleted form the current list of polygons), and thus they have been split into two pieces each (see polygons number 5f, 5b, 7f, 7b in Fig. 4). Notice that, in the example, the choice of the partition planes has been made to include also one intersecting case. In most cases, however, it is possible to choose the partition planes so that splitting of polygons is avoided. In this way, the construction process of the tree and the visit algorithm are faster; this is especially important when the BSP tree has to be built on line.[21]

3. PRE-SELECTION ALGORITHM

The accuracy of the estimate provided by the Kalman filter depends on the number of the available feature points. The inclusion of extra points will improve the estimation accuracy but will increase the computational cost. It has been shown that a number of five or six feature points, if properly chosen, may represent a good trade-off.[7] Automatic selection algorithms have been developed to find the optimal feature points.[17] In order to increase the computational efficiency of the selection algorithms, it is advisable to perform a pre-selection of the points that are visible to the camera at a given sample time. The pre-selection technique proposed in this paper is based on Binary Space Partitioning (BSP) trees.

Once that a BSP-tree representation of an object is available, it is possible to select the feature points of the object that can be visible from a given camera position and orientation by implementing a suitable visit algorithm of the tree. The algorithm can be applied recursively to all the nodes of the tree, starting from the root node, as detailed in the following Pascal-like procedure:

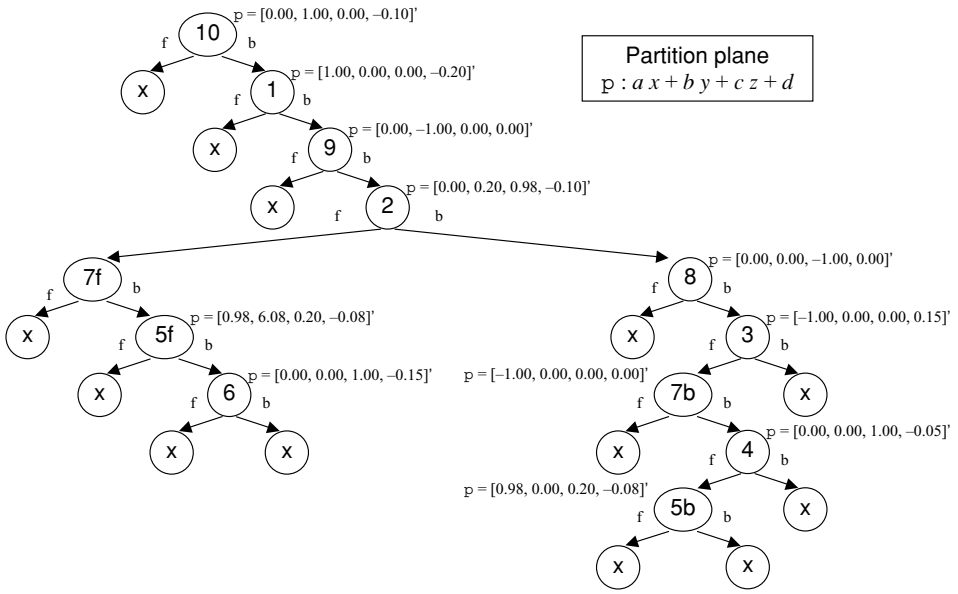

Fig. 3. BSP tree of the object shown in Fig. 2.

https://doi.org/10.1017/S0263574704001006 Published online by Cambridge University Press

```
procedure
pre-selection(node:BSP_tree;view:point;
    visible_points:point_list);
begin
    if node NOT EMPTY then
        case classify_point(view,node->partition_plane)
            ON_THE_PLANE:
            pre-selection(node->front_link,view,
                visible_points);
            pre-selection(node->back_link,view,
                visible_points);
            IN_FRONT_OF:
            pre-selection(node->back_link,view,
                visible_points);
            process_polygons(node->surfaces,view,
                visible_points);
            pre-selection(node->front_link,view,
                visible_points);
            IN_BACK_OF:
            pre-selection(node->front_link,view,
                visible_points);
            process_polygons(node->surfaces,view,
                visible_points);
            pre-selection(node->back_link,view,
                visible_points);
        end {case}
    end {if}
end {begin}
```

In the above procedure, the input variable `view` is the point of view (corresponding to the image plane of the camera) from which the current set of visible feature points of the object is evaluated. The visible points are listed in the variable `visible_points`, which contains also the projections of these points on the image plane of the camera.

The function `classify_point()` evaluates the position of the point of view with respect to the partition plane.

The core of the pre-selection algorithm is the procedure `process_polygons()`, which updates the current set of visible points by adding all the feature points of the polygons of the current node and by eliminating all the feature points that are hidden by the polygons of the current node.

The procedure is recursive and ends when all the nodes of the tree have been visited; at the end, the current set of visible points will contain all and only the feature points visible from the point of view. Notice that construction of the set proceeds so that the polygons are added in a sequence corresponding to their distance with respect to the point of view from the background to the foreground.

Notice that the code implementing the whole pre-selection algorithm (visit of the tree and polygons processing) exhibits a complexity $O(N)$, where N is the number of polygons of the object.[27] Moreover, some modifications can be introduced which allow a considerable reduction of the computational time. For example, all the polygons that are hidden with respect to the point of view (i.e., the angle between the normal vector to the polygon and the unit vector normal to the image plane is not in the interval $[-\pi/2, \pi/2]$ or the polygon is behind the image plane) can be discarded from the list and their feature points are not processed.

With reference to the BSP tree of Fig. 3, assuming that the point of view is placed as the observer of the image in Fig. 4, the visit sequence of nodes is: 10, 8, 7b, 4, 5b, 3, 2, 7f, 6, 5f, 9, 1; the polygons number 10, 8, 7b, 3, 7f result to be hidden and can be discarded from the list (i.e., their feature points are not processed).

The technique described above can be suitably exploited to set up a real-time pre-selection algorithm of the feature points on the camera image plane, using the prediction of the estimated pose of the target object provided by the Kalman filter.

4. SELECTION ALGORITHM

The pre-selection algorithm recognizes all the feature points that are visible from a camera view point. However, this does not ensure that all the visible points are "well" localizable, i.e., their positions can be effectively measured with a given accuracy. For instance, some points could be out of the image plane of the camera, or they could be too close each other to guarantee absence of ambiguity in the localization. Moreover, the number of the well localizable feature points may be larger than the optimal number of points ensuring the best pose estimation accuracy.

In the following, a windowing test is adopted to find the projections of the feature points that can be well localized. Then, a selection algorithm is used to choose an optimal subset of points to be considered for feature extraction.

4.1. Windowing test

The measurements of the coordinates of the projections of the feature points are obtained by considering suitable rectangular windows of the image plane to be grabbed and processed. Each window must contain one feature point. The windows are centered on the positions of the feature points on the image plane so as predicted by the Kalman filter. Their semi-dimensions are dynamically chosen in the interval $[Wr_{min}, Wr_{max}]$ for the base (the side parallel to the row's direction) and in the interval $[Wc_{min}, Wc_{max}]$ for the height (the side parallel to the column's direction). The minimum values are set so as to achieve a prescribed accuracy and robustness in the feature extraction, while the maximum values are set on the basis of the available memory and processing time.

A windowing test can be set up to select all the projections of the feature points that can be "well" localized.

First, all the points that are out of the field of view of the camera, or too close to the boundaries of the image plane, are discarded. This is achieved by eliminating all the points whose projections, so as predicted by the Kalman filter, are out of a central window of the image plane. The central window is obtained by reducing the height (base) of the whole image plane of the quantity Wr_{min} (Wc_{min}) from each side, as shown in Fig. 5.

Then, all the feature points that are too close to each other are discarded. This happens when the estimated distance between the projections of two or more points is lower than $S_f \cdot Wr_{min}$ ($S_f \cdot Wc_{min}$) along the row's (column's) direction; $S_f > 1$ is a suitable security factor.

All the remaining points are "well" localizable; the effective dimensions of the corresponding windows are dynamically adapted to the maximum allowable semi-dimension, so as to guarantee an assigned security distance

https://doi.org/10.1017/S0263574704001006 Published online by Cambridge University Press

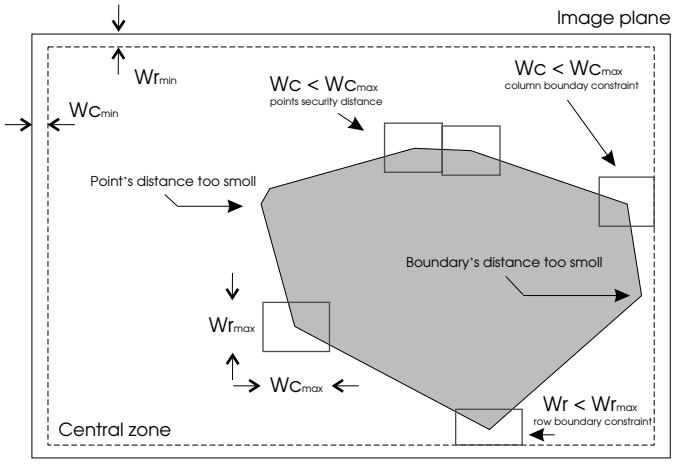

Fig. 5. Examples of significant situations during windowing test.

from the other points and from the boundaries of the image plane (see Fig. 5).

4.2. Optimal feature points selection

The number of feature points after pre-selection and windowing test may be larger than the optimal number of points (five or six) ensuring the best pose estimation accuracy.[7] The optimality of a given set of feature points can be valued through the composition of suitably selected quality indices into an optimal cost function. The quality indices must be able to provide accuracy, robustness and to minimize the oscillations in the pose estimation variables. To achieve this goal it is necessary to ensure an optimal spatial distribution of the projections of the feature points on the image plane and to avoid chattering events between different optimal subsets of feature points chosen during the object motion.

A first quality index is the measure of spatial distribution of the predicted projections on the image planes of a subset of n selected points:

$$Q_s = \frac{1}{n}\sum_{k=1}^{n} \min_{j\in\{1,\ldots,n\},\, j\neq k} \|\boldsymbol{p}_j - \boldsymbol{p}_k\|.$$

To ensure redundancy in case of faults of the feature extraction algorithm, n is chosen between 6 and 8.

A second quality index is the measure of angular distribution of the predicted projections on the image plane of a subset of n selected points:

$$Q_a = 1 - \sum_{k=1}^{n} \left| \frac{\alpha_k}{2\pi} - \frac{1}{n} \right|$$

where α_k is the angle between the vector $\boldsymbol{p}_{k+1} - \boldsymbol{p}_C$ and the vector $\boldsymbol{p}_k - \boldsymbol{p}_C$, being $\boldsymbol{p}_C$ the central gravity point of the whole subset of feature points, and the n points of the subset are considered in a counter-clockwise ordered sequence with respect to $\boldsymbol{p}_C$, with $\boldsymbol{p}_{n+1} = \boldsymbol{p}_1$.

To enhance the efficiency of the feature extraction process of the optimal subset of feature points and, hence, to increase the effective number of available points for the reconstruction, it is desirable to choose only those points with a good percentage of right extraction, with respect to each camera. Therefore, the third quality index measures the current *percentage of success* for the extraction process of a subset of q selected points for the camera:

$$Q_p = \prod_{k=1}^{q} \sigma_k$$

where $0 \leq \sigma_j \leq 1$ is the percentage of success for the extraction of the j-th feature point. These percentages have to be updated during the reconstruction process. At each sampling time and for all the feature points involved in the current extraction process, the percentage of the j-th point, for $j = 1, \ldots, q$, is updated by increasing (decreasing) the current value of a quantity $0 < \delta_p < 1$ in the case of a good (failed) extraction.

To avoid chattering phenomena, a quality index introducing hysteresis effects on the change of the optimal combination of points is considered:

$$Q_h = \begin{cases} 1+\epsilon & \text{if actual} = \text{previous combination} \\ 1 & \text{otherwise} \end{cases}$$

where ϵ is a positive constant.

The proposed indices are only some of the possible choices, but guarantee satisfactory performance when used with the pre-selection method and the windowing test presented in this paper. Other examples of quality indices are proposed, e.g., in reference [17].

The cost function is a simple product of the quality indices, but must be evaluated for all the possible combinations of the visible points on n positions. In order to perform a computationally efficient determination of the optimal set at each sample time, the initial optimal combination of points is first evaluated off-line; then, only the combinations that modify at most one point with respect to the current optimal combination are tested on-line, thus achieving a considerable reduction of processing time.

5. ESTIMATION PROCEDURE

A functional chart of the estimation procedure is reported in Fig. 6.

It is assumed that a BSP-tree representation of the object is built off-line from the CAD model.

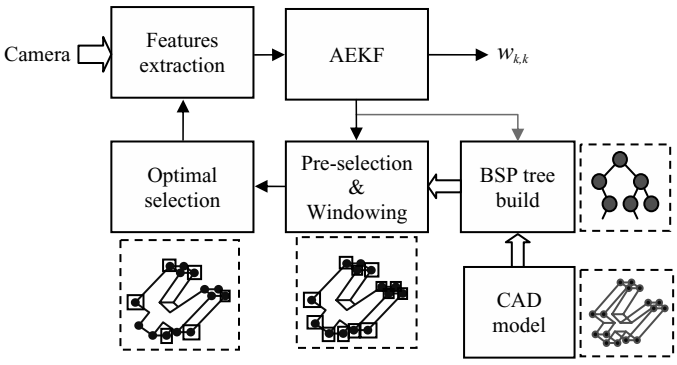

Fig. 6. Functional chart of the estimation procedure.

https://doi.org/10.1017/S0263574704001006 Published online by Cambridge University Press

A Kalman filter is used to estimate the object pose with respect to the base frame at the current sample time and to compute the predicted pose at next sample time.

Hence, the visit algorithm described in Section 3 is applied to the BSP tree of the object to find the set of all the feature points that are visible from the camera, when the object is in the predicted pose.

The resulting set of visible points is input to the selection algorithm described in Section 4 for the windowing test and optimal feature points selection.

At this point, the expected location of the optimal feature points in the image plane at the next sample time is computed and a dynamic windowing algorithm is executed to select the windows of the image plane to be input to the feature extraction algorithm.

Then, the image windows of the optimal selected points are elaborated using a feature extraction algorithm. Finally, the computed coordinates of the points in the image plane are input to the Kalman filter.

Notice that the procedure described above can be extended to the case of multiple objects moving among obstacles of known geometry[21]; if the obstacles are moving with respect to the base frame, the corresponding motion variables can be estimated using Kalman filters.

6. EXPERIMENTS

6.1. Experimental set-up

The experimental set-up is composed by a PC with Pentium IV 1.7GHz processor equipped with a MATROX Genesis board, a SONY 8500CE B/W camera, and a COMAU SMART3-S robot. The MATROX board is used as frame grabber and for a partial image processing (e.g. windows extraction from the image). The PC host is also used to realize the whole BSP structures management, the pre-selection algorithm, windows processing, the selection algorithm and the Kalman filtering. Some steps of image processing have been parallelized on the MATROX board and on the PC, so as to reduce computational time. The robot is used to move an object in the visual space of the camera; thus the object position and orientation with respect to the base frame of the robot can be computed from joint position measurements via the direct kinematic equation. The experimental set-up is shown in Fig. 7; notice that only one of the two cameras is used for the experiments.

In order to test the accuracy of the estimation provided by the Kalman filter, the camera was calibrated with respect to the base frame of the robot using the calibration procedure presented in reference [28], where the robot is used to place a calibration pattern in some known pose of the visible space of the camera. The camera resolution is 576×763 pixels and the nominal focal length of the lens is 16 mm, while the camera's calibration parameters are reported in Table II. Vector $\boldsymbol{\phi}_c$ contains the Roll, Pitch and Yaw angles of the camera frame with respect to the base frame, while the vector $\boldsymbol{c} = [g_1\ g_2\ g_3\ g_4\ d_1]$ contains the parameters used for compensating the distortion effects due to the imperfections of the lens profile and montage error alignment of the optical system, as described in reference [28]. The residual

Fig. 7. Experimental set-up.

calibration error on the plane perpendicular to the z_c-axis, at a distance of about 90 cm from the camera along the z_c-axis, is about 3.6 mm. The sampling time used for estimation is limited by the camera frame rate, which is about 26 fps. No particular illumination equipment has been used to test the robustness of the visual tracking system in the case of noisy visual measurements.

All the algorithms for BSP structure management, image processing and pose estimation have been implemented in ANSI C. The image features are the corners of the object, which can be extracted with high robustness in various environmental conditions. The feature extraction algorithm is based on Canny's method for edge detection[29] and on a simple custom implementation of a corner detector. In particular, to locate the position of a corner in a small window, all the straight segments are searched first, using an LSQ interpolator algorithm; then all the intersection points of these segments into the window are evaluated. The intersection points closer than a given threshold are considered as a unique average corner, due to the image noise. All the corners that are at a distance from the center of the window (which corresponds to the position of the corner so as predicted by the Kalman filter) greater than a maximum distance, are considered as fault measurements and are discarded. The maximum distance corresponds to the variance of the distance between the measured corner positions and those predicted by the Kalman filter.

Table II. Camera's calibration parameters resulting from the calibration procedure.

$r_0 = 273.51$
$c_0 = 353.64$
$f_u = -1963.87$
$f_v = 1960.32$
$\boldsymbol{o}_c = [\,-0.7985 \quad -1.1789 \quad -1.7598\,]^{\mathrm{T}}$ m
$\boldsymbol{\phi}_c = [\,90.204^\circ \quad 3.091^\circ \quad 91.570^\circ\,]^{\mathrm{T}}$
$\boldsymbol{c} = [\,2.323 \quad -0.6521 \quad 0.2506 \quad -16.70 \quad 154.5\,]^{\mathrm{T}} \cdot 10^{-3}$

https://doi.org/10.1017/S0263574704001006 Published online by Cambridge University Press

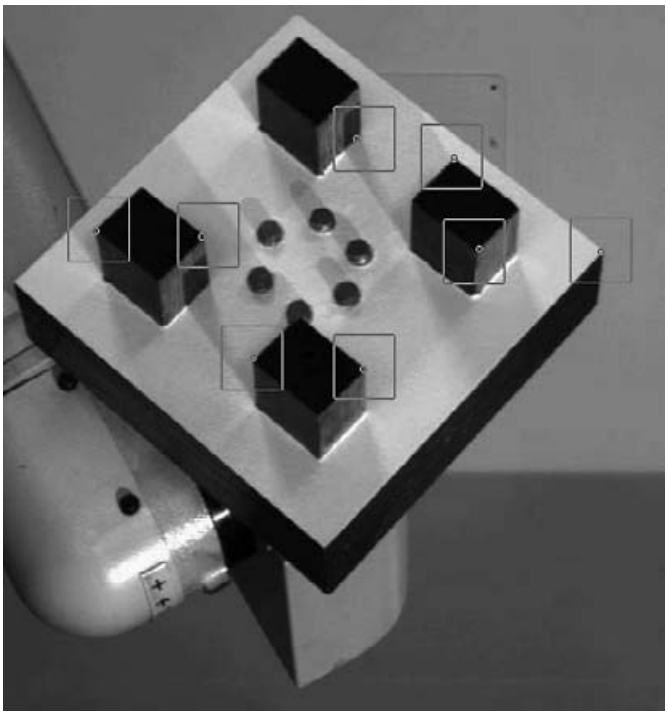

Fig. 8. Image seen by the camera.

The object used in the experiment (with 40 feature points) is shown in Fig. 8, so as seen from the camera during the motion. It is possible to recognize the windows selected for feature extraction as well as the measured positions of the feature points marked as points close to the center of each window.

6.2. *Experimental results*

In order to test the feasibility and the robustness of the proposed visual tracking system, two different experiments have been realized. The first experiment reflects a favorable situation where the object moves slowly and most of the feature points that are visible at the initial time remain visible during all the motion. The second experiment reflects an unfortunate situation where object moves quickly and the set of the visible points is very variable.

The time history of the trajectory used for the first experiment is represented in Fig. 9. The components are considered in the base frame. The maximum linear velocity is about 3 cm/s and the maximum angular velocity is about 3 deg/s.

The time history of the estimation errors is shown in Fig. 10. Noticeably, the accuracy of the system reaches the limit allowed by camera's calibration, for all the components of the motion, when the object does not move; during the motion the tracking errors grow but remain limited. As it was expected, the errors for some motion components are larger than others because only 2D information is available in a single camera system. In particular, the estimation accuracy is lower along the z_c axis for the position, and about the x_c and y_c axes for the orientation. Since, in the experiments, the z_c axis is almost aligned to the y axis of the base frame, the estimation errors are expected to be larger for the y component of the position as well as for the roll and yaw components of the orientation.

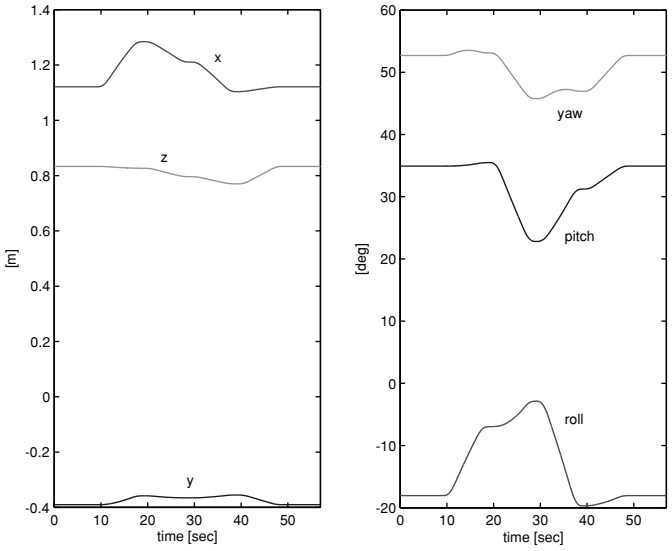

Fig. 9. Object trajectory with respect to the base frame used in the first experiment. Left: Position trajectory. Right: Orientation trajectory.

In Fig. 11 the output of the whole selection algorithm is reported. For each of the 40 feature points, two horizontal lines are considered: a point of the bottom line indicates that the feature point has been classified as visible by the pre-selection algorithm at a particular sample time; a point of the top line indicates that the visible feature point was chosen by the selection algorithm. Notice that 8 feature points are selected at each sample time, in order to guarantee at least five or six measurements in the case of fault of the extraction algorithm for some of the points. Also, some feature points are hidden during all the motion (points 4,12,20,28,36,40), some points are visible only over partial time intervals (points 1,3,7,8,9,17,25,33), while some points are visible and selected as optimal point during all the motion (points 2,16,21). Notice that no chattering phenomena are present, i.e. the frequency of the variations of the optimal set composition remains low.

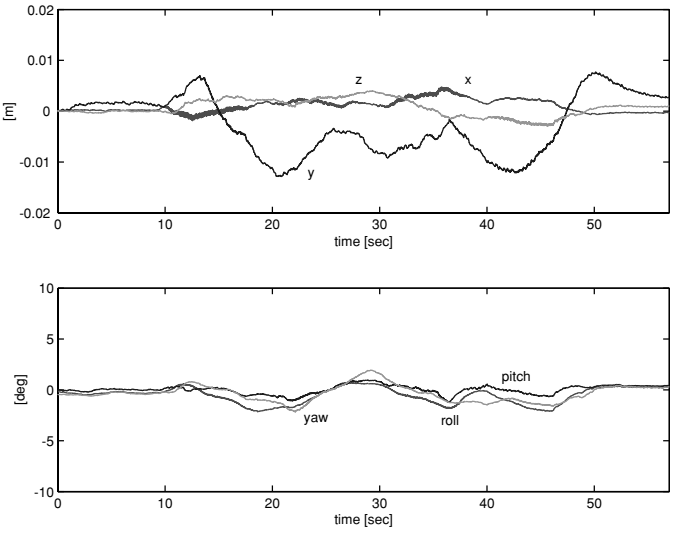

Fig. 10. Time history of the estimation errors in the first experiment. Top: Position errors. Bottom: Orientation errors.

https://doi.org/10.1017/S0263574704001006 Published online by Cambridge University Press

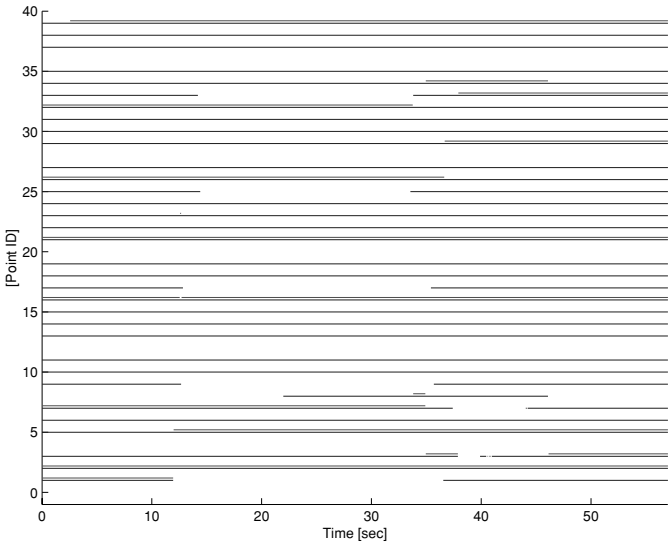

Fig. 11. Visible points and selected points in the first experiment. For each point, the bottom line indicates when it is visible, the top line indicates when it is selected for feature extraction.

The time history of the trajectory used for the second experiment is represented in Fig. 12. The maximum linear velocity is about 2 cm/s and the maximum angular velocity is about 7 deg/s.

The time history of the estimation error is shown in Fig. 13. It can be observed that the error remains low but is greater than the estimation error of the previous experiment. This is due to the increased velocity of the feature points and to the fact that from about $t = 15$ s to $t = 45$ s the object moves so that it is partially out of the visible space of the camera; also, it rotates in such a way that a side remains almost parallel to the image plane. This condition penalizes the estimation accuracy and explains why the magnitude of the estimation error components is greater than in the previous experiment. The corresponding output of the pre-selection and selection algorithms are reported in Fig. 14. It should be pointed out

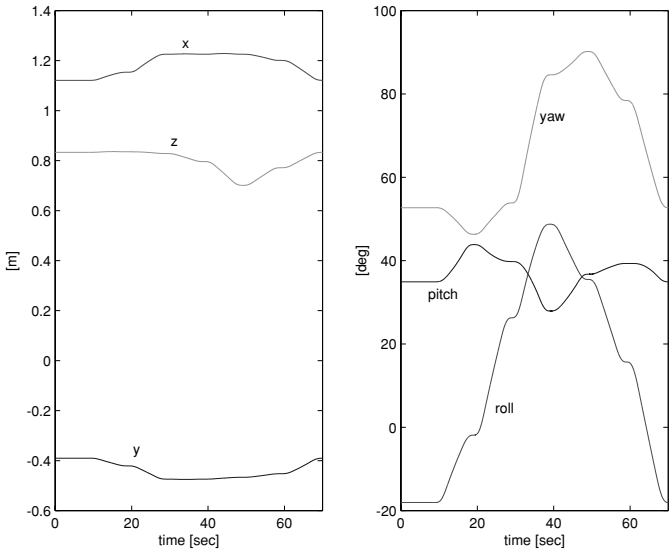

Fig. 12. Object trajectory with respect to the base frame used in the second experiment. Left: Position trajectory. Right: Orientation trajectory.

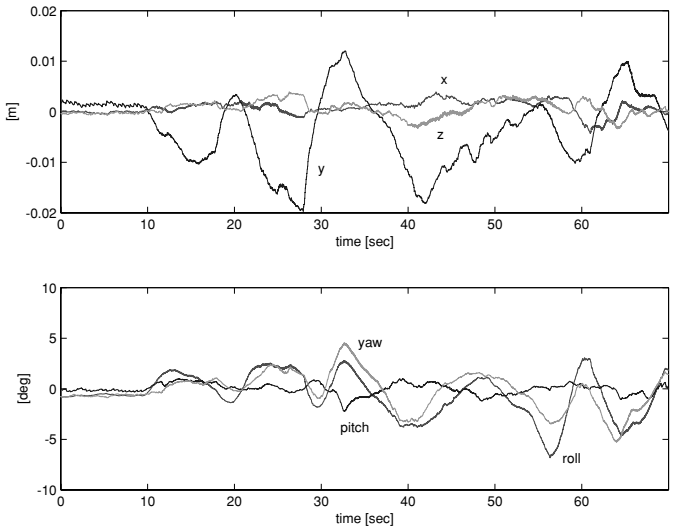

Fig. 13. Time history of the estimation errors in the second experiment. Top: Position errors. Bottom: Orientation errors.

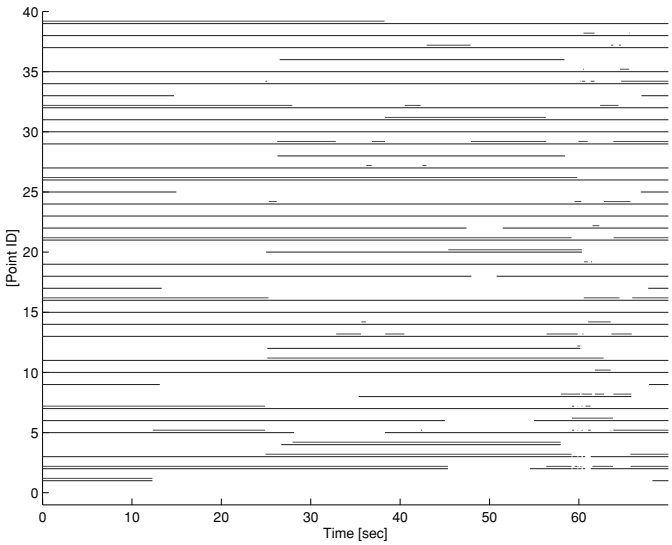

Fig. 14. Visible and selected points for the second experiment. For each points, the bottom line indicates when it is visible, the top line indicates when it is selected for feature extraction.

that the pre-selection and selection algorithm are able to provide the optimal set of points independently from the operating condition, although slight chattering phenomena appear in some situation where the elements in the set of visible points is rapidly changing.

7. CONCLUSION

The problem of real-time estimation of the pose (position and orientation) of a moving object from visual measurements was considered in this paper. The main contribution of the paper consists in a computationally efficient selection procedure that allows to evaluate the optimal set of feature points of the object to be used for image feature extraction and pose estimation. The procedure can be applied to polyhedral objects and is based on the representation of 3D objects by means of Binary Space Partitioning trees. The proposed algorithm is capable to find all the feature points of the object that are visibile to the camera at a given sample

https://doi.org/10.1017/S0263574704001006 Published online by Cambridge University Press

time with a computational complexity of order $O(N)$, where N is the number of polygons of the object. The estimation technique fully exploits the noise rejection and the prediction capabilities of the extended Kalman filter. The experimental results have confirmed the computational feasibility and the robustness of the proposed visual tracking scheme. Future work will be devoted at extending the selection procedure to the case of multiple target objects with interposing parts (e.g. the case of a robotic hand grasping a workpiece): in such a case a BSP tree model of the set of objects should be built and modified on-line.

Acknowledgments

This work was supported by Ministero della Istruzione, Università e Ricerca and Agenzia Spaziale Italiana.

References

1. S. Hutchinson, G. D. Hager and P. I. Corke, "A tutorial on visual servo control", *IEEE Trans. on Robotics and Automation* **12**, 651–670 (1996).
2. W. J. Wilson, "Relative end-effector control using cartesian position based visual servoing", *IEEE Trans. on Robotics and Automation* **12**, 684–696 (1996).
3. B. Espiau, F. Chaumette and P. Rives, "A new approach to visual servoing in robotics", *IEEE Trans. on Robotics and Automation* **8**, 313–326 (1992).
4. R. Horaud, F. Dornaika and B. Espiau, "Visually guided object grasping", *IEEE Trans. on Robotics and Automation* **14**, 525–532 (1998).
5. J. Baeten and J. De Schutter, "Hybrid vision/force control at corners in planar robotic-contour following", *IEEE/ASME Trans. on Mechatronics* **7**, 143–151 (2002).
6. S. Lee and Y. Kay, "An accurate estimation of 3-D position and orientation of a moving object for robot stereo vision: Kalman filter approach", *1990 IEEE Int. Conf. on Robotics and Automation* (1990) pp. 414–419.
7. J. Wang and W. J. Wilson, "3D relative position and orientation estimation using Kalman filter for robot control", *1992 IEEE Int. Conf. on Robotics and Automation* (1992) pp. 2638–2645.
8. J. N. Pan, Y. Q. Shi and C. Q. Shu, "A Kalman filter in motion analysis from stereo image sequence", *1994 IEEE Int. Conf. on Image Processing* (1994) pp. 63–67.
9. V. Lippiello, B. Siciliano and L. Villani, "Position and orientation estimation based on Kalman filtering of stereo images", *2001 IEEE Int. Conf. on Control Applications* (2001) pp. 702–707.
10. K. S. Arun, T. S. Huang and S. B. Blostein, "Least square fitting of two 3D point sets", *IEEE Trans. on Pattern Analysis and Machine Intelligence* **9**, 698–700 (1997).
11. B. K. P. Horn, K. M. Hilden and S. Negahdaripour, "Closed-form solution of absolute orientation using orthonormal matrices", *J. of Opt. Soc. Amer.* **A-5**, 1127–1135 (1998).
12. J. S.-C. Yuan, "A general photogrammetric method for determining object position and orientation", *IEEE Trans. on Robotics and Automation* **5**, 129–142 (1999).
13. K. Nickels and S. Hutchinson, "Weighting observations: The use of kinematic models in object tracking", *1998 IEEE Int. Conf. on Robotics and Automation* (1998) pp. 1677–1682.
14. F. Janabi-Sharifi and W. J. Wilson, "Automatic grasp planning for visual-vervo controlled robotic maniopulators", *IEEE Trans. on Systems, Man, and Cybernetics–Part B: Cybernetics* **28**, 693–711 (1998).
15. F. Keçeci and H.-H. Nagel, "Machine-vision-based estimation of pose and size parameters from a generic workpiece description", *2001 IEEE Int. Conf. on Robotics and Automation* (2001) pp. 2159–2164.
16. J. T. Feddema, C. S. G. Lee and O. R. Mitchell, "Weighted selection of image features for resolved rate visual feedback control", *IEEE Trans. on Robotics and Automation* **7**, 31–47 (1991).
17. F. Janabi-Sharifi and W. J. Wilson, "Automatic selection of image features for visual servoing", *IEEE Trans. on Robotics and Automation* **13**, 890–903 (1997).
18. K. Tarabanis, R. Y. Tsai and A. Kaul, "Computing occlusion-free viewpoints", *IEEE Trans. on Pattern Analysis and Machine Intelligence* **18**, 279–292 (1996).
19. P. C. Ho and W. Wang, "Occlusion culling using minimum occluder set and opacity map", *1999 IEEE Int. Conf. on Information Visualization* (1999) pp. 292–300.
20. T. W. Drummond and R. Cipolla, "Real-time tracking of complex structures with on-line camera calibration", *British Machine Vision Conf.* (1999) pp. 574–583.
21. V. Lippiello, B. Siciliano and L. Villani, "Objects motion estimation via BSP tree modeling and Kalman filtering of stereo images", *2002 IEEE Int. Conf. on Robotics and Automation* (2002) pp. 2968–2973.
22. C. Fagerer, D. Dickmanns and E. D. Dickmanns, "Visual grasping with long delay time on a free floating object in orbit", *Autonomous Robots* **1**, No. 1, 53–68 (1994).
23. K. Stark and S. Fuchs, "A method for tracking the pose of known 3-D objects based on an active contour model", *1996 IEEE Int. Conf. on Pattern Recognition* (1996) pp. 905–909.
24. P. Wunsch and G. Hirzinger, "Real-time visual tracking of 3-D objects with dynamic handling of occlusion", *1997 IEEE Int. Conf. on Robotics and Automation* (1997) pp. 2868–2872.
25. T. Drummond and R. Cipolla, "Real-time visual tracking of complex structures", *IEEE Trans. on Pattern Analysis and Machine Intelligence* **24**, 932–946 (2002).
26. V. Lippiello, B. Siciliano and L. Villani, "3-D objects motion estimation based on Kalman filter and BSP tree models for robot stereo vision", *Archives of Control Sciences* **12**, 71–88 (2002).
27. M. Paterson and F. Yao, "Efficient binary space partitions for hidden-surface removal and solid modeling", *Discrete and Computational Geometry* **5**, 485–503 (1990).
28. J. Weng, P. Cohen and M. Herniou, "Camera calibration with distortion models ad accuracy evaluation", *IEEE Trans. on Pattern Analysis and Machine Intelligence* **14**, 965–980 (1992).
29. J. Canny, "A Computational Approach to Edge Detection", *IEEE Trans. on Pattern Analysis and Machine Intelligence* **8**, 679–698 (1986).