

Visual Tracking of Multiple Objects Using Binary Space Partitioning Trees

Fabrizio Caccavale, Vincenzo Lippiello, Bruno Siciliano, and Luigi Villani

PRISMA Lab
Dipartimento di Informatica e Sistemistica
Università degli Studi di Napoli Federico II
Via Claudio 21, 80125 Napoli, Italy
<http://www.prisma.unina.it>

Abstract. The use of visual sensors may have high impact in robotic applications where it is required to measure the pose (position and orientation) and the visual features of objects moving in unstructured environments. In this paper, the problem of real-time estimation of the position and orientation of multiple objects is considered. Special emphasis is devoted to the case when two or more objects overlap with respect to the visual system causing occlusion. The algorithm is based on the Kalman filtering and Binary Space Partition (BSP) tree representations of the objects geometry. The real-time implementation of the algorithm is experimentally tested for the case of visual tracking of two objects using two cameras.

1 Introduction and State-of-Art

In the last decade the research on visual sensing has received a new impulse because digital signal processing hardware with high computational capability is becoming available at low cost. In fact, visual sensors offer the possibility to extract a great variety of information from a scene in a noninvasive manner, that can be used by automatic systems either at high level, e.g., for recognition and planning tasks, and at low-level, e.g., for autonomous guidance of vehicles, or robot visual servoing.

A typical problem in robotic vision is the real-time estimation of the position and orientation of moving objects of known geometry [1]. To this purpose, the visual system is in charge to measure some features of the objects, like the position of suitable points. Estimation accuracy grows when more cameras are used to guarantee different objects views and capture a larger number of feature points.

One of the major problems to cope with is represented by the noise affecting the visual measurements, due to temporal and spatial sampling and quantization of the image signal, lens distortion, etc., which may produce large estimation errors. The extended Kalman filter may improve the accuracy of visual tracking algorithms [2]; moreover, the prediction capability of the filter allows setting up a dynamic windowing technique of the image plane which may sensibly reduce processing time.

Another critical issue is that some feature points may be occluded to the visual system by the object itself (self occlusion) or may exit from the visual space, especially when a fixed-camera system is used. Moreover, during the execution of tasks like grasping or assembling, the feature points of an object may be occluded by another object (mutual occlusion).

In this paper, the extended Kalman filter is adopted to estimate the pose of moving objects using a fixed stereo camera system. The systematic procedure presented in [3] is used to compute the pose of the objects, combined to an efficient method of selection of the visible feature points [4]. This work is mainly focused on the selection procedure for multiple objects, based on Binary Space Partition (BSP) trees technique for real-time rendering of CAD models [5]. In detail, the prediction of the objects pose provided by Kalman filters is used to build a BSP tree of the whole system and to guide a visit of the tree which allows identifying all the feature points visible at the next sample time. Two different versions of the algorithm are presented: a simplified version that can be used for the case of objects with non interposing parts and a general version to be used for objects with interposing parts, e.g., a gripper grasping a tool. Complex situations can be handled in real time thanks to the computational low cost of BSP tree visit algorithms, whose complexity grows linearly with the number of feature points.

Particular emphasis is devoted to the presentation of experimental tests realized on a system of two cameras, for the case of two objects. The results confirm the effectiveness of the proposed approach.

2 Kalman filter

The position and orientation of a moving object can be estimated via an extended Kalman filter, using a system of n video cameras fixed in the workspace.

The object position and orientation with respect to a base frame is specified by defining a frame $O_o-x_o y_o z_o$ attached to the object and considering the coordinate vector of the origin $\mathbf{o}_o = [x_o \ y_o \ z_o]^T$ and the rotation matrix $\mathbf{R}_o(\boldsymbol{\varphi}_o)$, where $\boldsymbol{\varphi}_o = [\phi_o \ \alpha_o \ \psi_o]^T$ is the vector of the Roll, Pitch and Yaw angles.

Assuming that a CAD model of the object is available, m feature points are considered. The vectors $\mathbf{o}_o$ and $\boldsymbol{\varphi}_o$ can be computed from the measurements of the position of the feature points in the image planes of the cameras using the extended Kalman filter, based on a suitable model of the object motion. The recursive form of the equations for the problem at issue is reported in [3].

The accuracy of the estimate provided by the Kalman filter depends on the number of the available feature points. Inclusion of extra points improves the estimation accuracy but increases the computational cost. It has been shown that a number of five or six feature points, if properly chosen, represent a good trade-off [6]; to this purpose, suitable selection algorithms have been developed [7]. It should be pointed out, however, that the complexity of the selection algorithms grows at factorial rate. Hence, for objects with a large number of feature points, it is crucial to perform a pre-selection of the points that are visible to the cameras at each sample time.

3 Pre-selection algorithm

The pre-selection technique proposed in this paper is based on Binary Space Partitioning (BSP) trees. A BSP tree is a data structure that can be effectively adopted to

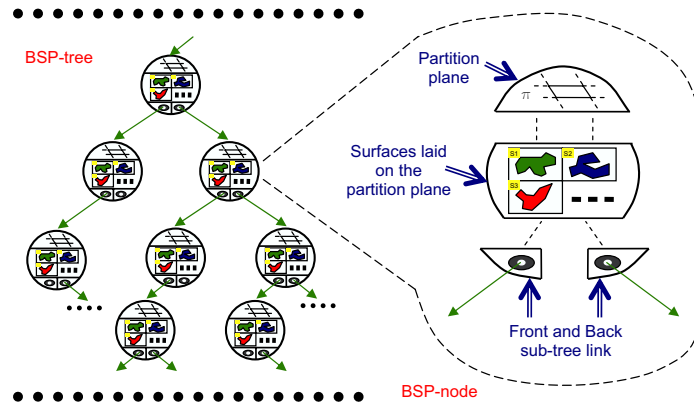


Fig. 1. BSP-tree structure and BSP-node composition.

represent the 3D CAD geometry of a set of objects [8]. In order to build the tree, each object has to be modelled as a set of planar polygons; each polygon is 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 one object is considered.

A node of the BSP tree is composed by four elements (see Figure 1): the partition plane, the set of surfaces laid 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, while the other is named back sub-space. To the purpose of this work, the partition plane must be chosen in the set of the planes containing the polygons corresponding to the object surfaces; hence, it 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.

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);
  {evaluate 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_surfaces,poly);
    IN_BACK_OF:
        add.to.list(back_surfaces,poly);
    SPANNING:
        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
end

```

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). Notice that a policy of choice of the partition plane can be imposed by suitably pre-sorting the list of polygons; the criteria may vary depending on the application of interest, e.g., achieve a balanced tree. To the purpose of this work, the partition planes have been chosen so that they do not intersect the remaining polygons of the list. The function `get_plane()` computes the equation of the partition plane for the current node. The function `classify_polygon()` determines 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. The procedure is recursive and ends when all the polygons and their parts are placed in a node of the tree.

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 recursive procedure:

```

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
end

```

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 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. Moreover, the code implementing the whole algorithm (visit of the tree and polygons processing) exhibits a complexity $O(N)$, where N is the number of polygons of the object [8].

4 Selection algorithm

The pre-selection algorithm does not guarantee that all the visible points are “well” localizable, i.e., their positions can be effectively measured with a given accuracy. Moreover, the number of the well localizable feature points may be larger than the optimal number of points ensuring the best pose estimation accuracy.

To this purpose, a windowing test is adopted first to select the projections of the feature points that can be well localized. Then, an optimal set of feature points is identified by using an optimal cost function based on a combination of suitable quality indices. These 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. Moreover, in order to exploit the potentialities of a multi-camera system, it is important to achieve an optimal distribution of the feature points among the different cameras. In this work, the following indices are considered:

- Spatial distribution index:

$$Q_{si} = \frac{1}{q_i} \sum_{k=1}^{q_i} \min_{\substack{j \in \{1, \dots, q_i\} \\ j \neq k}} \|p_j - p_k\|,$$

where q_i is a subset of selected points for the i -th camera, $i = 1, 2$.

- Angular distribution index:

$$Q_{ai} = 1 - \sum_{k=1}^{q_i} \left| \frac{\alpha_k}{2\pi} - \frac{1}{q_i} \right|,$$

where q_i is a subset of selected points for the i -th camera, $i = 1, 2$, α_k is the angle between the vector $\mathbf{p}_{k+1} - \mathbf{p}_{C_i}$ and the vector $\mathbf{p}_k - \mathbf{p}_{C_i}$, being $\mathbf{p}_{C_i}$ the central gravity point of the whole subset of feature points. The q_i points of the subset are considered in a counter-clockwise ordered sequence with respect to $\mathbf{p}_{C_i}$, with $\mathbf{p}_{q_i+1} = \mathbf{p}_1$.

- Anti-chattering index:

$$Q_h = \begin{cases} 1 + \epsilon & \text{if } \Gamma = \Gamma_{opt} \\ 1 & \text{otherwise} \end{cases}$$

where ϵ is a positive constant and Γ (Γ_{opt}) is the optimal set of points at the actual (previous) sample time.

- Point distribution indices:

$$Q_e = 1 + \frac{2}{q} \left(\frac{2}{q} - 1 \right) \left| q_1 - \frac{q}{2} \right|$$

$$Q_d = \frac{q_1/d_1 + q_2/d_2}{q / \min\{d_1, d_2\}},$$

where q_i is the number of points assigned to the i -th camera, and d_i is the distance of the i -th camera from the object, $i = 1, 2$. The first index ensures an equal distribution of points among the cameras; the second index takes into account the distance of the cameras from the object.

The cost function is chosen as

$$Q = Q_h \frac{Q_e Q_d}{q} (q_1 Q_{s1} Q_{a1} + q_2 Q_{s2} Q_{a2})$$

and must be evaluated for all the possible combinations of the visible points on q positions. To determine 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 for camera with respect to the current optimal combination are tested on line, thus achieving a considerable reduction of processing time.

5 Visual tracking of multiple objects

The visual tracking algorithm for the case of a single moving object is reported and experimentally tested in [4]. The estimation procedure assumes that a BSP tree representation of the object is built off-line from the CAD model. This procedure can be extended to the case of multiple moving objects of known geometry. In fact, the algorithm can handle in an effective way both self occlusions and mutual occlusions. To this purpose, two different situations shall be considered: the case of objects whose parts cannot be interposed, and the case of objects with interposing parts (e.g., a gripper grasping an object).

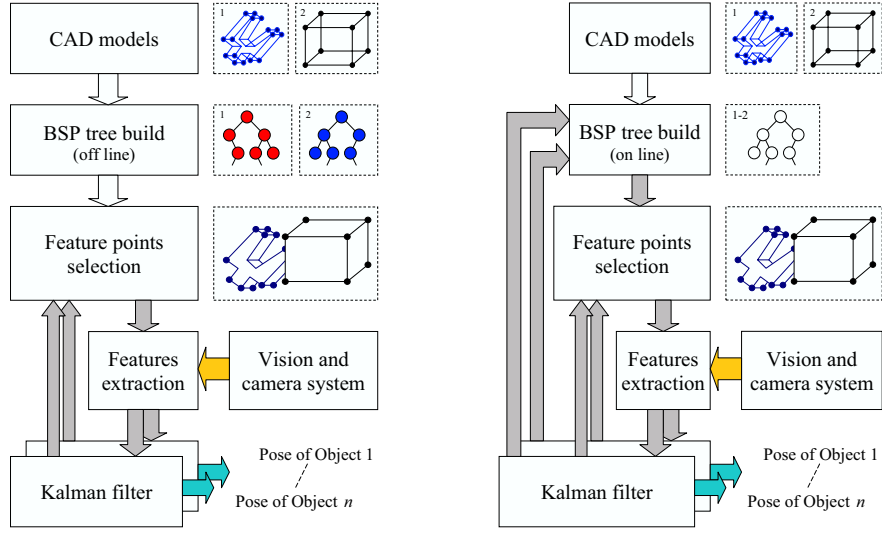


Fig. 2. Estimation procedure. Left: non interposing objects. Right: interposing objects.

In the first case (see Fig. 2 (left)), it is assumed that a BSP tree representation of each object is built off-line from the CAD model. A different Kalman filter is required for each object to estimate the corresponding pose with respect to the base frame at the next sample time. The procedure for feature points selection has to be modified as follows:

- **Step 1:** For each camera, an ordered sequence of BSP trees is built, according to the estimated distance of the objects from the camera, from the farthest to the nearest.
- **Step 2:** For each camera, a visit algorithm is applied to each BSP tree of the sequence to find the set of all the feature points that are visible from the camera. In particular, for each BSP tree of the sequence, a current set of visible points is updated, by discarding the feature points of the previous objects occluded by the current object and by adding the visible feature points of the current object.
- **Step 3:** For each object, the resulting set of visible points is input to an algorithm for the selection of the optimal set of feature points Γ_{opt} for the n -camera system.
- **Step 4:** For each camera and each object, the location of the optimal feature points in the image plane at the next sample time is computed.
- **Step 5:** For each camera, a dynamic windowing algorithm is applied to select the parts of the image plane to be input to the feature extraction algorithm.

At this point, all the image windows of the optimal selected points are elaborated using a feature extraction algorithm. The computed coordinates of the points in the image plane, for each object, are input to the corresponding Kalman filter which provides the estimate of the actual object pose and the predicted pose at the next sample time used in Steps 1-2.

In the case of multiple objects with interposing parts, the above procedure may fail because the objects cannot be correctly ordered with respect to the distance from

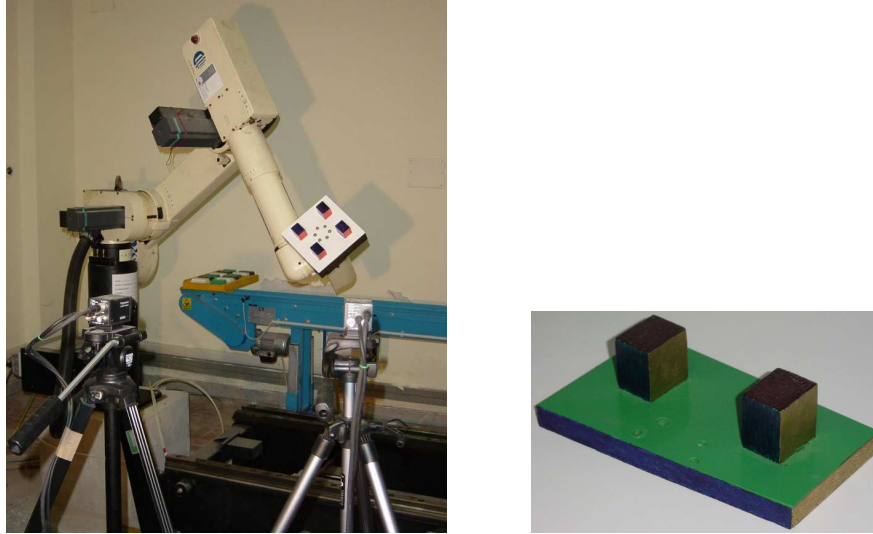


Fig. 3. COMAU Smart3-S robot carrying the moving object (left); fixed object (right).

the camera. This problem can be overcome, at the expense of computation time though, by adopting the solution represented in the functional chart of Fig. 2 (right). As before, a Kalman filter is used for each object to estimate the corresponding pose with respect to the base frame at the next sample time. Differently from the previous case, a unique BSP tree representation of all the objects is built on line, using the CAD model and the estimation provided by the Kalman filters. Hence, for each camera, the visit algorithm of the tree is applied once to find the set of all the visible points. Then, Steps 3, 4, and 5 follow. The procedures described above can be applied also to the case of objects moving among obstacles of known geometry.

6 Experiment

The experimental set-up is composed by a PC with Pentium IV 1.7GHz processor equipped with two MATROX Genesis boards, two SONY 8500CE B/W cameras, and a COMAU Smart3-S robot. The robot is used to move one object in the visual space of the cameras (see Fig. 3 left), while the object on the right of Fig. 3 is placed on a known position of the workspace; hence the “true” position and the orientation of the moving object, with respect to the base frame of the robot, can be computed from joint position measurements via the direct kinematic equation. The feature points are the corners of the objects, which can be extracted with high robustness in various environmental conditions; the object carried by the robot has 40 corners while the other has 26 corners. The cameras (disposed as in Fig. 3) have been calibrated with respect to the base frame of the robot. The resolution of each camera is 576×763 pixels and the nominal focal length of the lenses is 16 mm. The estimation sampling time is limited by the camera frame rate, which is about 26 fps.

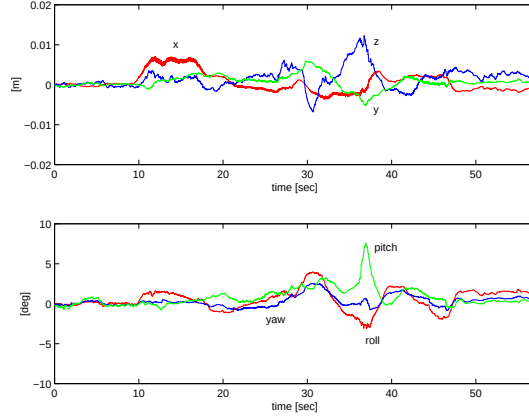


Fig. 4. Estimation errors for the moving object. Top: Position. Bottom: Orientation.

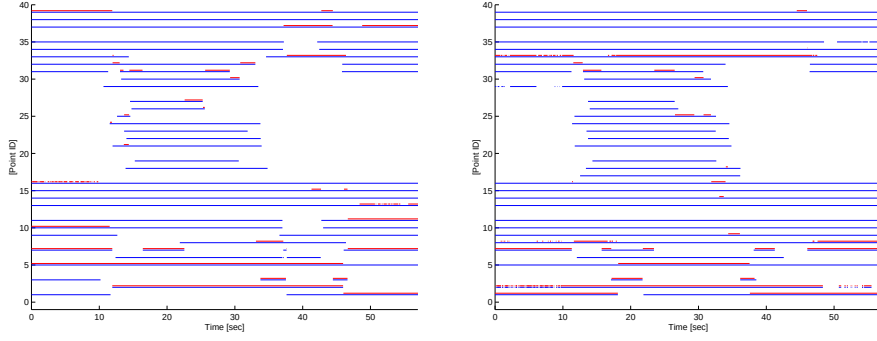


Fig. 5. Visible and selected points of the moving object for camera 1 (left) and 2 (right).

To evaluate the capability of the proposed algorithm in coping with occlusions, an experimental test has been realized where the fixed object temporarily occludes the moving object. The trajectory of the moving object has a maximum linear velocity of about 3 cm/s and a maximum angular velocity of about 3 deg/s; the fixed object (see Fig. 3 left) is placed in the pose ($\mathbf{o}_o = [1.102 \quad -0.504 \quad 0.794]^T \text{m}$, $\varphi_o = [-85.7 \quad 59.2 \quad 48.7]^T \text{deg}$).

The time history of the estimation errors for the moving object is shown in Fig. 4. Notice that the error is small also in the presence of occlusions; moreover, the errors components remain of the same order of magnitude, thanks to the use of two cameras. In Fig. 5 the output of the selection algorithm, for the two cameras, is reported. For each of the 40 feature points of the moving object, two horizontal lines are considered: a point on the bottom line indicates that the feature point was classified as visible by the pre-selection algorithm at a particular sample time; a point on the top line indicates that the visible feature point was chosen by the selection algorithm. In Fig. 6 the total number of the visible, selected and localized points of the moving object are shown. Notice that $\sum_i q_i = 8$ feature points are selected

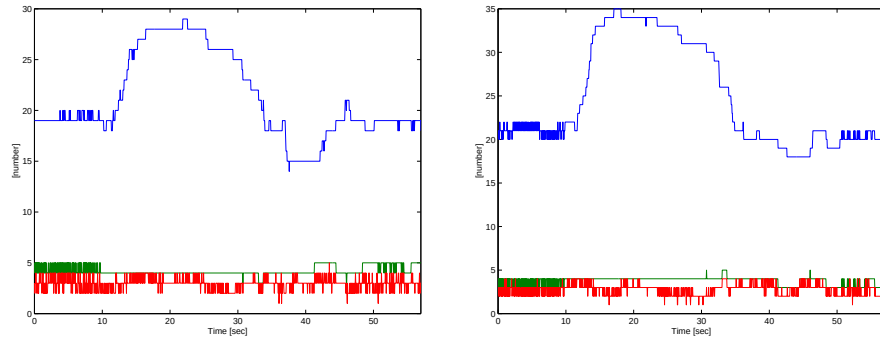


Fig. 6. Number of visible (blue), selected (green) and localized (red) points of the moving object for camera 1 (left) and 2 (right).

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. Both Figs. 5 and 6 evidence the presence of occlusions in the first and in the last part of the experiment.

7 Conclusion and Research Perspective

The algorithm presented in this paper may represent a starting point to solve an important open issue for robotic applications: the visual tracking of objects in an unstructured and dynamic environments. A typical application may be the grasping of a moving object guided by a fixed visual system. For this scenario, the proposed methodology may be used to develop a new strategy of automatic detection of the occlusions that happen during the grasp execution, which can improve the task reliability. Similar problems may arise in cooperative robots applications.

References

1. S. Hutchinson, G.D. Hager, and P.I. Corke, "A tutorial on visual servo control," *IEEE Trans. on Robotics and Automation*, vol. 12, pp. 651–670, 1996.
2. J.W. Wilson, C.W. Hulls, and G. Bell, "Relative end-effector control using cartesian position based visual servoing," *IEEE Trans. on Rob. and Aut.*, vol. 12, pp. 684–696, 1996.
3. V. Lippiello, B. Siciliano, and L. Villani, "Position and orientation estimation based on Kalman filtering of stereo images," *IEEE Conf. on Control Applications*, pp. 702–707, 2001.
4. V. Lippiello, B. Siciliano, and L. Villani, "Objects motion estimation via BSP tree modeling and Kalman filtering of stereo images," *IEEE Conf. on Rob. and Aut.*, pp. 2968–2973, 2002.
5. T.W. Drummond and R. Cipolla, "Real-time tracking of complex structures with on-line camera calibration," *British Machine Vision Conf.*, pp. 574–583, 1999.
6. J. Wang and J.W. Wilson, "3D relative position and orientation estimation using Kalman filter for robot control," *IEEE Conf. on Robotics and Automation*, pp. 2638–2645, 1992.
7. F. Janabi-Sharifi and W.J. Wilson, "Automatic selection of image features for visual servoing," *IEEE Trans. on Robotics and Automation*, vol. 13, pp. 890–903, 1997.
8. 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.