Providing Guidance for Maintenance Operations Using Automatic Markerless Augmented Reality System

Hugo Álvarez† Iker Aguinaga‡ Diego Borro‡
CEIT and Tecnun (University of Navarra), Spain

Abstract

This paper proposes a new real-time Augmented Reality based tool to help in disassembly for maintenance operations. This tool provides workers with augmented instructions to perform maintenance tasks more efficiently. Our prototype is a complete framework characterized by its capability to automatically generate all the necessary data from input based on untextured 3D triangle meshes, without requiring additional user intervention. An automatic offline stage extracts the basic geometric features. These are used during the online stage to compute the camera pose from a monocular image. Thus, we can handle the usual textureless 3D models used in industrial applications. A self-supplied and robust markerless tracking system that combines an edge tracker, a point based tracker and a 3D particle filter has also been designed to continuously update the camera pose. Our framework incorporates an automatic path-planning module. During the offline stage, the assembly/disassembly sequence is automatically deduced from the 3D model geometry. This information is used to generate the disassembly instructions for workers.


1 INTRODUCTION

Augmented Reality (AR) is a set of technologies that enrich the way in which users experience the real world by embedding virtual objects in it that coexist and interact with real objects. This way, the user can be exposed to different environments and sensations in a safe and more realistic way. AR has been applied successfully to many areas, such as medicine [7], maintenance and repair [9], cultural heritage [8] and education [17].

To embed virtual objects in the real world, the camera pose must be known. Moreover, the techniques that achieve this goal can be classified into two groups: marker tracking [5] and markerless tracking [9]. While the first group requires environment adaptation, since some patterns are added to the scene, the second option uses natural features that are already in the scene. Besides, the markerless tracking generally needs the user intervention to make an unpleasant training process.

In this paper, we describe a real-time automatic markerless AR disassembler for maintenance and repair operations. The system generates instructions indicating which the next step is and the way to proceed. The instructions are represented graphically by superimposing a virtual representation of the next step on top of the real view of the system being repaired. Thus, the next part to disassemble is virtually coloured in red and translated along the extraction path (Figure 8). Indeed, a virtual arrow emphasize the extraction direction. The online assistance provided can, therefore, reduce the time spent by technicians searching for information in paper-based documentation, and thus, it enables faster and more reliable operation.

The main challenge addressed by this work is the automatic (without user intervention) generation of the suitable information to provide user feedback. The system receives as an input a single untextured 3D triangle mesh of each component. The assembly/disassembly sequence is computed automatically from this input, finding collision-free extraction trajectories and the precedence relationship of the disassembly of the different parts composing the target system [2]. Additionally, some geometric features, such as edges and junctions, are also automatically extracted to identify and track the different components during their manipulation [3].

The rest of the paper is structured as follows. Section 2 presents the state of the art on the use of AR for assembly/disassembly tasks. Section 3 describes in detail our AR disassembler. Section 4 presents the results and tests performed on our prototype system. Finally, Section 5 presents the conclusions reached in this work and future research lines.

2 RELATED WORK

Several works [30, 38] demonstrate the benefits of AR based approaches applied to the assistance in maintenance and repair operations in terms of operation efficiency. Compared to Virtual Reality (VR), AR offers a more realistic experience because the user interacts with real objects, while VR manipulates virtual objects and is limited by the lack of suitable sensors feedback. Further, VR is oriented to training rather than guidance, improving the skills of users through multiple simulations, while AR can be used for both training and guidance. Because of that, many authors have been addressed the problem of building an AR tool that provides guidance for maintenance operations. The main characteristics of some of these solutions are detailed below.

[45] describes an AR application for assembly guidance using a virtual interactive tool. The user controls different instruction messages with a pen and the assembly steps are displayed in the image. However, instructions are limited to 2D photographic images on the camera image border. The system does not perform 3D model tracking to offer 3D augmentation data. Similarly, [4] describes an AR maintenance tool based on an untracked system.

Other approaches [5, 10, 13, 24, 34, 41, 46] rely on marker based tracking systems to compute the camera position and orientation. Thus, instructions are registered in the real world with a high degree of realism. All these systems require the physical modification of the environment by adding patterns that are readily detectable in the image. These patterns can simply be a piece of paper [5, 10, 24, 41, 46], or more sophisticated infrared markers [13, 34]. They achieve high accuracy with low computational cost. However, the drawbacks of this kind of system are well known. They require environment adaptation, which is not always possible, and marker
and 3D model coordinate systems must be properly calibrated by the user. In addition, marker tracking systems can fail when the markers are partially occluded. This is a likely scenario in maintenance operations, since occlusions with the hands of the worker and tools are common. This problem is minimized by some authors using a multi camera system [13, 34] or a multi marker system [18].

Other alternatives use natural features to retrieve the 6 DOF of the camera. During an offline process, some views of the scenario are taken. These keyframes are used as input for a training processes, where 2D-3D correspondences are established by back-projecting detected 2D image features into the known 3D model. During the online phase, 2D features are extracted from the camera image and matched to those that were indexed in the offline phase, thus, obtaining the camera pose. Using this procedure, [9] describes an application for airplane maintenance, while [27] details a prototype for car repair guidance. These solutions do not modify the environment, but they need some user intervention to generate the training datasets. Furthermore, they are optimized for textured scenes, which do not always exist in industrial environments.

Some interesting solutions do not need the 3D model, but require high interactivity with users [31]. Some virtual annotations (disassembly instructions) are added manually to the scene by clicking on the image position that should be. This way, the system estimates the 3D location of the notes automatically and renders them correctly registered to the environment using SLAM techniques.

Compared to all these works, our approach proposes a more robust framework. It is able to compute automatically the assembly/disassembly sequence, which has not been addressed by any of the papers cited above, since this information is usually considered an input parameter given by the user. Additionally, our markerless tracking is based on untextured 3D model geometry, and all the tracking data is extracted automatically, without user intervention. This lets us overcome the problems outlined earlier: environment adaptation and textureless scenes.

### 3 Automatic Markerless AR Disassembler

Figure 1 shows all the steps of the proposed framework. The automatic offline phase receives as input a 3D model of the system to maintain. This model is composed of several parts, each one with its own 3D triangle mesh and located in its assembly position (Figure 1). With this minimal information, the disassembly-planning module is able to extract the precedence order in which the parts should be placed or removed (Section 3.1). Furthermore, some geometric features, such as edges and junctions, are extracted to address the tracking problem (Section 3.2.1).

These geometric features let us perform the 3D model recognition (Section 3.2.2) and tracking (Section 3.2.3) during the online phase. Within this phase, once the camera’s 6 DOF have been computed, the system offers an augmented instruction, indicating how to assemble/disassemble the next part. Then, the users perform the corresponding operation and notify the system that they want to continue with the next step. Currently this notification is implemented as a simple keystroke, but it could be implemented with a more sophisticated voice command. Each step of this process is explained in depth in the next sections.

#### 3.1 Disassembly Planning

The assembly or disassembly procedure for a product can be described by two main components: a precedence graph and the disassembly paths for each component [22]. The precedence graph describes the order of the operations as a graph. In this graph each node represents a component and it is connected to a set of other elements whose removal must precede the disassembly of the element represented by it. The second component, the disassembly path, represents the motion that is required to extract the component from the assembly.

The initial works on the generation of assembly and disassembly sequences started at the end of the 80’s and early 90’s [15]. The works on disassembly planning can be classified in two main groups (component based and product based planners) according to the input information used. In component based planning, this information is based on the description of the components of the product by means of its geometry, behaviour, type, etc. The algorithm is usually divided into two sub-problems: the localization of a direction which allows the local translation of the components and a second step validates this extraction direction by checking if the generated path is collision free. Some of the first works were based on the recursive partitioning of the assembly into subassemblies [42]. The second class are product based planners, which use more abstract input describing the product, as the precedence relationships of the disassembly process. Their objective is the generation of optimal sequences. For this purpose these approaches used optimization algorithms such as Genetic Algorithms [26].

Due to the objectives of this work, we use a component based planner [1], whose outcome is the precedence graph and disassembly path for a system. Once this information is computed, the assembly/disassembly sequences can be obtained by traversing the graph from a target component, and removing each other node connected to it.

#### 3.1.1 Model Format

Input 3D models are composed of a set of parts that must be disassembled. The 3D model of Figure 1, for example, is composed by 7 parts: a box, a lid, four small screws, and a cord connector similar to a big screw.
No texture information is required as input because it is limited to the geometry of the 3D model. Each part of the model is defined as a simple 3D triangle mesh, and all of them must be provided with respect to the same origin and located in their initial configuration, i.e., as if the model was assembled. It is noteworthy that although CAD software usually allows defining explicitly some relationships between the geometric position of the different components of a system, they are not required for our system.

3.1.2 Precedence Graph and Disassembly Path
To obtain the precedence graph and disassembly paths, we proceed heuristically to select different components, and then, we try to generate a disassembly path for them. For most components in industrial settings the extraction path of any component can be represented by a single straight line. We use the contact information of the different components in order to find the set of free local extraction directions. Afterwards, we test the different directions for collisions in the extraction. Unfortunately, this procedure fails when the extraction path is more complex. In this case, we use a more robust path planning algorithm based on Rapid Growing Random Trees (RRT) [2]. This approach is more flexible but also more expensive computationally (see Section 4).

If an extraction path is found for a component, we can find which components need to be previously removed by testing the removal path with a set of all the components in their initial configuration. Any component that collides with positions in the extraction path needs to be removed prior to it. This way, we can build the complete precedence graph in a trial and error process.

Notice that the assembly/disassembly sequence graph is computed only once for each input 3D model. The entire process is executed in a few seconds during the offline stage. This sequence is used by the Recognition and Tracking module to figure out which is the following scenario to recognize and track (Section 3.2).

3.2 Recognition and Tracking
The markerless tracking algorithm is divided into two stages: automatic offline processing and online tracking. In the offline stage, some geometric features such as edges and junctions, are extracted automatically from the input 3D triangle mesh. In addition, a set of synthetic views of these geometric features are indexed in a database. This stage is executed only once per model. During the online stage, this data is used to calculate the camera pose. This stage consists of two different problems - recognition and tracking. The recognition problem tries to compute the camera pose with previous knowledge. The input camera image is matched to one of the synthetic views generated during the offline stage. Thus, it is used for both initialization and tracking failure recovery. The tracking problem, in turn, assumes that the information from the previous frame is available. It performs a frame-to-frame tracking of the 3D model edges extracted during the offline stage.

An input 3D model is composed of several components. Therefore, the presence or removal of a part modifies the geometric composition of the model (Figure 2). Because of this, for each assembly/disassembly step detected by the automatic disassembly-planning module, all the processes explained above are executed. Each geometric configuration that results from a part removal is treated as a separate 3D model, for which geometric features are extracted. Notice that although multiple disassembly sequences can be considered, only the sequential order set by the disassembly-planning module is used to get the collection of possible geometric configurations. In the same way, during the online execution the next step is known, and it is clear which geometric configuration should be loaded, since the sequence has a predefined order. This method increases the memory requirements, but it offers robust recognition and tracking results.

3.2.1 Geometric Feature Extraction
Given an input 3D triangle mesh (without texture) our geometric feature extraction module detects automatically (without user intervention) 3D sharp edges, and junctions. Meanwhile the 3D sharp edges extraction has been inspired by [16, 44], as far as we know, the 3D junctions extraction has not previously been addressed.

Figure 2: Geometric features for different geometric configurations.

3D sharp edges are defined as edges of neighbour facets whose normal angle is larger than a predefined threshold (40 degrees in our experiments). Moreover, a junction is defined as a point where several sharp edges meet. This way, if two sharp edges share one vertex, and the angle that they form is close to 90 degrees, then a 3D junction is built. These types of junctions are known as L junctions. However, this technique also considers sharp edges and junctions that belong to the inner boundary of the model (Figure 3). These types of edges and junctions are always hidden by other parts of the model, and do not provide relevant information. Similarly, undesired sharp edges and junctions can be detected due to bad tessellation. Thus, a refinement step that uses the OpenGL pipeline has been developed to discard these outlier edges and junctions. This step is similar to that explained in [28], but rather than using a set of 2D views to build 3D edges, we proceed in reverse order, validating the quality of an initial estimation of 3D edges through a set of 2D views. Therefore, the model is rendered from different points of view, following a sphere path. Good results have been obtained with steps of altitude and azimuth of 30 degrees, which gives a total of 144 views. For each view, a 2D edge map is built, applying the sobel operator to the depth buffer image that is generated by OpenGL when the 3D model triangle mesh is rendered, similar to [43]. This 2D edge map stores the projection of strong edges and junctions that are visible from the current point of view. On the other hand, visible 3D edges and junctions are extracted for the current view from the collection of 3D sharp edges and junctions using the OpenGL occlusion query. In a case where a 3D sharp edge is visible and its projection appears in the 2D edge map, it receives a vote. This vote scheme is analogous for junctions and repeated for all views. Finally, only the 3D sharp edges and junctions with a minimum number of votes are retained as inliers, since they are strong 3D edges and junctions that belong to the outer boundary. We have noticed that this refinement procedure is a necessary step to guarantee the robustness of the recognition and tracking procedures.

Figure 3: 3D sharp edges before and after refinement.
3.2.2 Recognition

3D model recognition in monocular image consists of recovering the camera pose that defines the viewpoint from which the model is shown in the image. This task is made more difficult by the lack of knowledge about the previous frames. Several authors use an offline learning stage to solve this problem [9, 14, 27], where a set of keyframes are obtained to build a database of 2D-3D feature visual cues. Nevertheless, they require user intervention to get the training dataset, and some of them do not work well in textureless scenes. Because of this, we have developed a recognition method that automatically builds the training dataset, rendering the virtual 3D model from multiple points of view. It uses geometric properties of an input 3D model to get positive correspondences during the recognition process, similar to [20, 35, 36], which allow handling textureless environments. Compared to [20], both approaches use multiple camera pose hypotheses and a voting scheme based on geometric properties to select the correct pose, but our recognition method does not require the attachment of an inclination sensor to the camera, which simplifies the problem to 4 unknowns (camera azimuth and position) and reduces its application domain.

In case of our recognition module, the only input parameters used are the intrinsic camera matrix and the geometric features extracted in the previous section. These 3D geometric features are matched to 2D image features, obtaining a set of potential camera poses, which are weighted using a contour similarity measure. Finally, the camera pose with highest score is returned. A deeper explanation of this method can be found in [3], but we give a brief explanation here for the completeness of the paper.

During the automatic offline learning stage, several 2D synthetic views of these 3D features are generated moving a virtual camera along a sphere. Thus, rotation ranges and steps for each axis are parameters that must be set. They can be set by the user or inferred from the disassembly-planning module. Since the next part of assembly/disassembly is known, we can train only the views that put this component in front of the camera, ruling out other views. This is not a strict requirement, since the user will usually try to focus the camera on this part as an instinctive behaviour. Moreover, this action improves the memory requirements and the computational cost, since there will be fewer keyframes to process. Assuming that the next part to process is put in front of the camera (with all the model completely visible and covering the entire image), good results have been obtained processing only those views that are in the [-45, 45] range for all axes. Additionally, angle steps that vary from 5 to 8 degrees have been used, which gives a total of 2000-5000 keyframes, depending on the complexity of the model and the maximum storage capacity.

Then, each virtual keyframe is processed individually. For each virtual keyframe the projection of visible 3D sharp edges and junctions is stored. Notice that the visibility test is determined by a virtual keyframe the projection of visible 3D sharp edges and gives a total of 2000-5000 keyframes, depending on the complexity of the model and the maximum storage capacity.

3.2.3 Tracking

Given the intrinsic camera parameters and the data generated in the previous frame, the frame-to-frame tracking implemented for this work computes the extrinsic camera parameters, based on temporal coherence and the combination of multiple visual cues. See [11] for a detailed explanation about tracking methods that use different visual cues.
The surface of the model, we only retain a small subset of them barycentric coordinates. Although many features can appear on features that lie on the surface of the model are back-projected using Then, assuming that the pose of the previous frame is known, features from the previous frame using the FAST operator [32].

3D point tracking

We apply the best properties of the point based trackers presented edge tracker with a robust feature tracker. It is well known that motions, several authors [32, 39] have integrated an accurate 3D point cloud to deal with appearing and disappearing points due in general to rotations. During the 3D point tracking step, features selected by the 3D point generation step are tracked with KLT [25]. This provides an estimation of the current 2D position of these 3D points. To get the correct new 2D location, the current image is searched for a feature with a similar SIFT descriptor. Taking into account the temporal coherence between consecutive frames, this search is limited to the vicinity of each feature (a window search of 75 pixels for our experiments), while the matching of two SIFT descriptors follows the simple rule mentioned in [6]: two descriptors are matched if the euclidean distance of the second nearest descriptor is significantly greater than the distance to the nearest descriptor. If no match is found, this 2D-3D correspondence is discarded. Finally, the new camera pose is updated with the remaining 2D-3D correspondences. Using the previous camera pose as an initial guess, a non-linear optimization of the point reprojection error is performed. More precisely, the pseudo-Huber robust cost function [27] is minimized to reduce the influence of outlier correspondences.

Particle Filter

The combination of the edge tracker and point based tracker presented above offers good tracking results. However, the point based tracker is only available when a minimum set of features are detected on the surface of the model. This requirement is generally satisfied, since the placement or removal of parts favours the presence of corners. An example of this property can be observed in Figure 8, where assembled screws generate multiple salient points on the surface of the Box-1 model. Notice that this is not the case for some models (Matryoshka model in Figure 8), where not enough features are detected due to the homogeneity of the outer surface. Thus, we have incorporated a 3D particle filter that uses multiple cues to handle these cases. Particle filters are a robust method against non-static scenes in which multi-modality is likely [29]. They are divided into two stages: particle generation and particle evaluation. In the particle generation step, the previous camera pose is perturbed to generate multiple camera pose candidates (particles). Thus, the posterior density of the motion parameters is approximated by a set of particles. Then, the particle evaluation step is responsible for assigning a weight to each particle and selecting the correct one. In our experiments, particle generation uses a random walk motion model, while the particle evaluation uses the 3D junctions and edges extracted in the offline stage and the 3D points reconstructed by the point based tracker. The likelihood (w) of each particle (P_i) is inversely proportional to the reprojection error of visible 3D junction centers (X_i) and back-projected 3D features (Y_i), taking into account how many pixels of the projection of visible 3D edges coincide with an image edge:

\[ w(P_i) = \frac{|V_i|}{Z_i} \left( \sum_{j=1}^{m} C(\delta(P_i, X_j)) + \sum_{j=1}^{m} C(\delta(P_i, Y_j)) \right) \]  

where |Z_i| is the number of pixels that results from the projection of 3D visible edges with the camera matrix P_i provided by the

Edge Tracker

The 3D edges extracted in the offline stage are used to refine the camera pose of the current frame, similar to [39, 43]. Visible 3D edges are detected and projected into the current camera image using the OpenGL occlusion query and the previous camera pose. Then, for each projected edge a set of 2D edge samples are generated. Notice that the 3D coordinates of these samples are known. A local search along the edge normal is done to establish the presence of these samples in the current camera image. If an edge pixel is detected in the vicinity of an edge sample, then a 2D-3D correspondence is found. This procedure can provide multiple match hypotheses for each edge sample, which are handled with the robust Tukey estimator. A non-linear optimisation of the Tukey reprojection error is applied to get the final camera pose of the current frame.

Feature Tracker

The recursive edge tracking presented above is fast and precise, but it does not support fast movements. To cope with abrupt camera motions, several authors [32, 39] have integrated an accurate 3D edge tracker with a robust feature tracker. It is well known that point based trackers can deal with rapid motions, since features are relatively easy to localize and match between frames. Therefore, we apply the best properties of the point based trackers presented in [6, 27]. Optical flow estimates the new feature locations, while the SIFT descriptor is used to get a more accurate position. It can be divided into two main steps: 3D point generation and 3D point tracking. The 3D point generation step extracts the 2D features from the previous frame using the FAST operator [32]. Then, assuming that the pose of the previous frame is known, features that lie on the surface of the model are back-projected using barycentric coordinates. Although many features can appear on the surface of the model, we only retain a small subset of them due to real-time requirements (30 features in our experiments). This way, the features with strongest response are selected first. Moreover, for each selected feature the simplified version of the SIFT descriptor [33] is stored. Compared to the original SIFT descriptor, this version uses a fixed size patch (25 pixels for our experiments) and only the strongest orientation is returned. These modifications, combined with parallel techniques, offer a reduction in computational cost at the expense of robustness. Nonetheless, the resulting robustness is enough for our tracking purposes, and the computational cost is decreased considerably, satisfying real time requirements (see Section 4). Additionally, the 3D point generation step is executed in every frame, continuously refreshing the 3D point cloud to deal with appearing and disappearing points due in general to rotations. During the 3D point tracking step, features selected by the 3D point generation step are tracked with KLT [25]. This provides an estimation of the current 2D position of these 3D points. To get the correct new 2D location, the current image is searched for a feature with a similar SIFT descriptor. Taking into account the temporal coherence between consecutive frames, this search is limited to the vicinity of each feature (a window search of 75 pixels for our experiments), while the matching of two SIFT descriptors follows the simple rule mentioned in [6]: two descriptors are matched if the euclidean distance of the second nearest descriptor is significantly greater than the distance to the nearest descriptor. If no match is found, this 2D-3D correspondence is discarded. Finally, the new camera pose is updated with the remaining 2D-3D correspondences. Using the previous camera pose as an initial guess, a non-linear optimization of the point reprojection error is performed. More precisely, the pseudo-Huber robust cost function [27] is minimized to reduce the influence of outlier correspondences.

Particle Filter

The combination of the edge tracker and point based tracker presented above offers good tracking results. However, the point based tracker is only available when a minimum set of features are detected on the surface of the model. This requirement is generally satisfied, since the placement or removal of parts favours the presence of corners. An example of this property can be observed in Figure 8, where assembled screws generate multiple salient points on the surface of the Box-1 model. Notice that this is not the case for some models (Matryoshka model in Figure 8), where not enough features are detected due to the homogeneity of the outer surface. Thus, we have incorporated a 3D particle filter that uses multiple cues to handle these cases. Particle filters are a robust method against non-static scenes in which multi-modality is likely [29]. They are divided into two stages: particle generation and particle evaluation. In the particle generation step, the previous camera pose is perturbed to generate multiple camera pose candidates (particles). Thus, the posterior density of the motion parameters is approximated by a set of particles. Then, the particle evaluation step is responsible for assigning a weight to each particle and selecting the correct one. In our experiments, particle generation uses a random walk motion model, while the particle evaluation uses the 3D junctions and edges extracted in the offline stage and the 3D points reconstructed by the point based tracker. The likelihood (w) of each particle (P_i) is inversely proportional to the reprojection error of visible 3D junction centers (X_i) and back-projected 3D features (Y_i), taking into account how many pixels of the projection of visible 3D edges coincide with an image edge:

\[ w(P_i) = \frac{|V_i|}{Z_i} \left( \sum_{j=1}^{m} C(\delta(P_i, X_j)) + \sum_{j=1}^{m} C(\delta(P_i, Y_j)) \right) \]  

where |Z_i| is the number of pixels that results from the projection of 3D visible edges with the camera matrix P_i provided by the

Figure 5: 3D Recognition pipeline during the online execution.
particle \( i \), \( |V_i| \) is the number of \( Z_i \) pixels whose distance to the closest image edge point is below a predefined threshold, \( \delta \) defines the reprojection error, \( C \) represents the pseudo-Huber cost function (with a threshold of \( \sim 10 \) pixels for our experiments), \( n \) is the number of visible 3D junction centres and \( m \) is the number of back-projected 3D features.

To calculate the reprojection error, the current 2D location of the corresponding points must be known. Thus, to get the 2D location of the 3D junction centres in the current frame, those that were visible in the previous frame are tracked using KLT. Notice that the current 2D position of back-projected 3D features has been tracked previously by the point based tracker, and it is not calculated again by the 3D particle filter. All these 2D translation estimations are also used to fix the optimum number of particles in accordance with demand (between 200 and 1000 in our experiments). If strong 2D translations are detected, the number of particles is increased, while in the absence of movement, few particles are needed.

Efficient distance transforms are used to obtain the ratio of inlier edge pixels \( (|V_i|/|Z_i|) \), similar to [19], but executing all the calculations in the CPU. Visible 3D edges are projected, and for each 2D edge sample, the distance to the closest edge pixel is obtained with a simple look-up in the image distance transform. Thus, \( Z_i \) is the set of all 2D edge samples, while \( V_i \) is the set of 2D edge samples whose distance to the closest edge pixel is below a predefined threshold. Furthermore, we consider the orientation of edges to discard false matches [12]. Thus, the current camera edge image is split into four orientation specific edge images, and for each edge image its corresponding distance transform is computed. In this way, an input 2D edge sample is only compared to the image distance transform indexed by its orientation.

Further, to reduce the computational cost, visible 3D edges and junctions are pre-computed for a set of points of views along a 3D sphere. This increases memory requirements, but decreases the execution time. In addition, a particle annealing is applied to get correct results with a limited number of particles [29].

### Integration of Multiple Trackers

The overall markerless tracking scheme is shown in Figure 6. The point based tracker is executed first to handle strong movements. If a minimum number of 3D points are back-projected (7 for our experiments), then the 3D point tracking step is executed and the camera pose is updated. Otherwise the 3D particle filter is called to approximate the current camera pose. Afterwards, the edge tracker is executed to refine the camera pose, avoiding the possible error caused by a non-accurate reconstruction of the 3D point cloud or non-accurate particle likelihood calculation. The point based tracker and the 3D particle filter offer a good initial guess of the correct camera pose, while the edge tracker calculates a precise one. It should be noted that both the point based tracker and the particle filter are executed in down-sample resolution images to reduce the computation time. The edge tracker, meanwhile, is executed at the original scale to get more accuracy (Section 4). This procedure is executed every frame while the edge tracker works correctly. To determine the success of the edge tracker, the similarity measure explained in [37] is used. Thus, visible 3D edges are projected using the camera pose refined by the edge tracker, and the score is the sum of the dot products between the gradients of these projected samples and the image edge gradients at the corresponding locations. If this score is below a predefined threshold (0.7 for our experiments), then the edge tracker fails, and the system starts the recovery mode and returns to the recognition stage (Section 3.2.2).

### Experiments

In this section we present the results of the proposed AR disassembler. The hardware setup consists of an Intel Core i7-860 at 2.80GHz and 3GB of RAM equipped with a Logitech QuickCam Connect webcam.

In Table 1 the execution time of the Disassembly-Planning module (it includes disassembly graph as well as the disassembly paths and sequences) is presented for the set of 3D models shown in Figure 8. It is capable of obtaining the disassembly sequence of parts with different geometric properties in a few seconds. It can also find complex disassembly paths using the RRT algorithm (Section 3.1), as is required for the Gear-box example.

<table>
<thead>
<tr>
<th>Model</th>
<th>Disassembly steps</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box-1</td>
<td>6</td>
<td>1.5</td>
</tr>
<tr>
<td>Box-2</td>
<td>3</td>
<td>0.69</td>
</tr>
<tr>
<td>Matryoshka</td>
<td>5</td>
<td>1.6</td>
</tr>
<tr>
<td>Gear-box</td>
<td>2</td>
<td>101.660 (RRT)</td>
</tr>
<tr>
<td>Elephant</td>
<td>20</td>
<td>6.09</td>
</tr>
</tbody>
</table>

Table 2 displays the output of the Geometric Feature Extraction module for different geometric configurations.

The response of the 3D recognition module against the geometric configurations processed in Table 2 is presented in Table 3. A set of keyframes were taken for each model using a marker tracking system [40] to obtain these values. Thus, the real camera pose is known for these keyframes, and it provides us with the mechanism to compute the accuracy of the recognition. The automatic training of the 3D recognition module uses a range of \([-45,45]\) degrees applied for each axis, starting from the point of view that put the next part to disassemble in front of the camera (Section 3.2.2). All test were executed at 640x480 resolution, finding the first camera pose with high accuracy in a few hundreds milliseconds.

To prove the benefits of our tracking method and compare the different tracking techniques we use, we have executed the same video sequence (640x480 resolution) with three different tracking configurations. The first one is the tracker presented in the previous Section, combining a point based tracker, a particle filter and an edge tracker (Point + PF + Edges). The second one disables the particle filter (Point + Edges), and the last one disables the point based tracker (PF + Edges). The main goal of this comparison is
to show the advantages of using different types of tracking methods together, obtaining a more robust tracking and satisfying the real time requirements. The point based tracker and the particle filter perform their calculations in down-sampled resolution (320x240) to reduce the computational time, while the edge tracker is executed in the original scale (640x480) to achieve high accuracy. Additionally, the particle filter is limited to a maximum set of 1000 particles to control the total runtime. Similarly to the 3D recognition experiment, we have used a marker tracking system to obtain the real camera pose of each video frame. Furthermore, we have repeated all the process with three different 3D models (Box-1, Matryoshka and Gear-box) to simulate multiple conditions. We have chosen these three models because they offer us the majority of possible scenarios. The Box-1 model represents an industrial model with some corners on its surface, the Matryoshka model represents a model with an homogeneous outer surface, without feature presence, and the Gear-box represents an industrial model with complex geometry. The results of these 3D models for the video sequence (provided with the paper) are shown in Figure 7. It is noteworthy that there is an increase in measurement error because

Table 2: Execution time for the Geometric Feature Extraction. The number of facets measures the complexity of the extraction procedure.

<table>
<thead>
<tr>
<th>Model</th>
<th>Facets</th>
<th>Edges</th>
<th>L Junctions</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box-1</td>
<td>3102</td>
<td>118</td>
<td>24</td>
<td>1.43</td>
</tr>
<tr>
<td>Box-2 (step 3)</td>
<td>40</td>
<td>21</td>
<td>32</td>
<td>1.33</td>
</tr>
<tr>
<td>Matryoshka</td>
<td>208</td>
<td>12</td>
<td>24</td>
<td>1.47</td>
</tr>
<tr>
<td>Gear-box</td>
<td>59658</td>
<td>172</td>
<td>59</td>
<td>26.84</td>
</tr>
<tr>
<td>Elephant</td>
<td>9084</td>
<td>72</td>
<td>80</td>
<td>3.65</td>
</tr>
</tbody>
</table>

Table 3: Accuracy and execution time for recognition.

<table>
<thead>
<tr>
<th>Model</th>
<th>Mean Reprojection Error (pixels)</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box-1</td>
<td>4.26</td>
<td>99.66</td>
</tr>
<tr>
<td>Box-2 (step 3)</td>
<td>4.47</td>
<td>63.16</td>
</tr>
<tr>
<td>Matryoshka</td>
<td>4.55</td>
<td>107.52</td>
</tr>
<tr>
<td>Gear-box</td>
<td>6.57</td>
<td>155.03</td>
</tr>
<tr>
<td>Elephant</td>
<td>4.72</td>
<td>356.05</td>
</tr>
</tbody>
</table>

the video sequence consists of rapid camera movements that try to lose the tracking, and the non-perfect calibration between the marker and model reference systems. Box-1 and Gear-box are tracked successfully with the three tracking configurations, while the Point+Edges option fails for the Matryoshka example after frame 110, due to high reprojection error. The reason for this failure is that the Matryoshka model has an homogeneous outer surface where not many features are detected. Thus, in the absence of features the point based tracker fails, and the particle filter is the more robust tracker. Nevertheless, in the Box-1 example, where many features lie on the surface of the box due to the presence of screws, the point based tracker runs faster and has lower reprojection error than the particle filter. For the Gear-box example, although the point based tracker is slower than the particle filter, its reprojection error is lower. In addition, we have observed during the experiments that the point based tracker can track fast camera movements that the particle filter option can not. Thus, since it is very common to detect a minimum set of features on the surface of the model, the point based tracker is the best option for most examples. A tracker that combines both of them gets the best properties of each option, running fast and accurately for models with features on their surface, and offering high robustness in absence of features.

The execution time of Box-1, Matryoshka and Gear-box models for the video sequence using (Point + PF + Edges) configuration is detailed in Table 4. It displays the execution time required by each tracking step, including the Image Processing step, which handles a large camera image (640x480). Although the Point Tracker step includes the computation of the SIFT descriptors parametrized with 4x4 regions and 8 orientations (4x4x8=128 bins), its execution time is low because of its simplifications (Section 3.2.3). Notice that the total execution time is kept below real time limits.

Table 4: Execution time of each tracking step of the (Point + PF + Edges) configuration for Box-1, Matryoshka and Gear-box models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
</tr>
<tr>
<td>Box-1</td>
<td>7.29</td>
</tr>
<tr>
<td>Box-2 (step 3)</td>
<td>12.14</td>
</tr>
<tr>
<td>Matryoshka</td>
<td>0.78</td>
</tr>
<tr>
<td>Gear-box</td>
<td>2.47</td>
</tr>
<tr>
<td>Elephant</td>
<td>22.73</td>
</tr>
</tbody>
</table>

In Figure 8, some snapshots of augmented instructions that our system provides are shown for the set of 3D models presented in Table 2. This set of models covers multiple geometric configurations, different types of surfaces, as well as disassembly steps of varying complexity. Figure 8 (a)-(f) shows the good response of our system against models with specular surfaces. In addition, Figure 8 (i) displays a satisfactory output when small parts are tracked. Figure 8 (j)-(n) demonstrates the robustness of the system for scenes where multiple parts with similar geometry are put in front of the camera. In fact, this also proves that the system will not be jeopardized by the tools that the worker places in the task-space, provided that objects and tools do not have the same shape. Likewise, the recognition and tracking of Box-2 and Matryoshka models is hampered by their textureless and homogeneous outer surface. Figure 8 (p) indicates that our tracking can handle models with high complexity, combining both rectilinear and curvilinear edges. Notice that although the extraction path of the gear is composed of several movements in multiple directions (it can be clearly appreciated in the video file
We have also observed some limitations in the recognition module. As the recognition of the model processes the entire image, complex scenarios with cluttered backgrounds may cause the detection of some false positives. This is noticeable when the user points to a keyboard, which has numerous edges, and the presence of an object is identified due to a positive edge matching. If this happens, a simple camera shake will be enough to lose the tracking and restart the recognition. Note that the tracking module can handle cluttered backgrounds because it combines multiple visual cues (points, edges and junctions) and uses temporal coherence assumptions, which imposes restrictions on the search space. In order to reduce false positives of recognition, high score matching thresholds have been set, which discard hypotheses with poor matching scores, but requires that the model should be almost completely visible to get a positive detection. Another option would be to impose a restriction on color similarity to validate the detection, which should be set during the offline phase. On the other hand, the recognition module relies on the existence of edges and junctions in the model, so its application domain is reduced to polyhedral objects. However, the system is oriented to textureless models, so if the model has no appreciable texture or geometric properties, then it would be difficult to address the recognition problem. Therefore, the inclusion of a module that handles objects with texture and no geometric properties would extend the domain of our system, making it fairly generic.

5 Conclusions and Future Work

This paper presents an automatic AR disassembler oriented to maintenance and repair operations. The novelty of our system relies on its ability of being a complete framework that is self-supplied. All the data is extracted automatically from a collection of 3D parts that are provided as untextured 3D triangle meshes. In addition, a Disassembly-Planning module is responsible for computing the assembly/disassembly sequence, which is obtained finding collision-free trajectories. Moreover, a 3D recognition module based on geometric properties, such as junctions and edges, retrieves the first camera pose of the input untextured 3D model. Thus, our AR disassembler can deal with scenes that lack texture. Furthermore, a robust markerless tracking that uses multiple techniques has been developed. This module combines an edge tracker, a point based tracker and a 3D particle filter to offer a robust 3D tracking against multiple and undesirable conditions, such as homogeneous surfaces. The proposed AR disassembler is a complete framework that runs in real-time and tries to facilitate the work of workers, replacing the paper-based documentation.

A visual inspection that ensures the correctness of the assembly/disassembly task and provides guidance feedback is a problem that will be addressed by the authors in the future. A GPU implementation of the 3D particle filter and a set of user experiments to validate the usability of the system will also be explored further.

Acknowledgements

The contract of Hugo Álvarez is funded by the Government of the Basque Country within the Formación de Personal Investigador program that belongs to the Education, University and Research department.

References

Figure 8: Examples of augmented instructions that our system displays for a set of different 3D models. Instructions are in red, while the 3D model that is being tracked is highlighted in green.