Model-Based 3D Visual Tracking of Rigid Bodies using Distance Transform

Marios Loizou

Department of Computer Science
University of Cyprus
1 Panepistimiou Avenue, Aglantzia
Nicosia, Cyprus
Email: mloizo11@cs.ucy.ac.cy

Paris Kaimakis

School of Sciences
University of Central Lancashire Cyprus
12 - 14 University Avenue, Pyla
Larnaka, Cyprus
Email: pkaimakis@uclan.ac.uk

***Abstract*—The core idea of model-based 3D tracking is that of continuously estimating the pose parameters of a 3D object throughout a sequence of images, e.g., a video feed. Here, we present an edge-based method for achieving 3D object tracking, via Gauss-Newton optimization. We rely on natural features observations, like edges, for the detection of interest points and by using the 3D pose of the object in the previous frame, we correctly estimate its new 3D position and orientation, in real-time. There is also a C++ implementation of the visual tracking system, with the use of the OpenCV library, which can be found in our GitHub repository (https://github.com/marios2019/Visual_Tracking).**

Keywords–Object 3D tracking; Model-based; Gauss-Newton optimization; Distance Transform

I. Introduction and Related Work

Object 3D tracking is used in a variety of Computer Vision applications, like Augmented Reality [1] where virtual objects are super-imposed to the scene and in Robot Object Manipulation [2] [3] where the target object is manipulated with the use of a mechanical device. Also, 3D tracking enables cultural heritage reconstruction applications, where usually through a mobile device the user can reanimate and view ancient architecture. In all of these cases, the goal is to estimate the 3D pose (position and orientation) of the object with respect to the observer.

There are many approaches to 3D tracking, depending on the targeted application and the means that are being used [4] [5]. Techniques like [6] and [7], fall into the marker-based tracking category, were they make use of point and planar markers, that are carefully placed in the scene by the user. Because of their pattern uniqueness, they can be identified as image features, which lead to 2D-3D correspondences with high precision. The latter provides reliable measurements for pose estimation. Despite their good performance, marker-based tracking techniques require engineering the environment, which sometimes the application's end-users dislike and sometimes is impossible, e.g., outdoor environments. By contrast, 3D tracking by detection techniques, are based on natural features that can be detected in the scene. Works like [8], construct a database from Scale-invariant Feature Transform (SIFT) features [9], that are detected from images with different viewpoints of the object. Multi-view correspondences can be found and the 3D positions of the features are recovered using Structure-from-Motion (SfM) algorithms. At runtime, SIFT features are extracted for each video frame, which yield to 2D-3D correspondences. Camera pose can be estimated using algorithms like Random Sample Consensus (RANSAC) [10]. Also recent advances in Deep Learning has given rise to techniques like [11] and [12] for simultaneously detecting and tracking multiple objects, although they require huge datasets for training. In this work we are focused on non-learning methods, as they do not required collecting, analysing and preprocessing huge amount of data.

Most of the work that is been done belongs to feature-based 3D tracking category, where camera pose estimation, just like tracking by detection, relies on natural features, like edges or corners. Furthermore, techniques that belong to this category provide a strong prior knowledge of the camera pose for each new frame, which aids the pose estimation task. This yields to a jitter-free camera pose between consecutive frames, unlike tracking by detection methods, where camera pose is recovered in each frame indepentedly. Edge-based methods like Real-time Attitude and Position Determination (RAPiD) [13] or [14], sample the edges of the model into 3D control points, which they are rendered and impose onto the image, along with model. Each control point is matched with a point that lies on a detected edge, by searching along the normal of the edge that the control point belongs to. Given enough control points, pose estimation is achieved, by minimizing the sum of squares of the perpendicular distances, using a least squares approach.

Our approach on model-based 3D visual tracking belongs to the feature-based 3D tracking category, specifically in the edge-based methods, as we rely on measurements being made along an edge, to find the displacement between the virtual and real object. Unlike RAPiD methods, the measurements are not being made on 3D edges of the virtual model, but on their 2D projections on the image. Additionally, we do not search along the normal direction of each edge, to find correspondences between control points and points on the detected edges. Instead, we detect features - edges from the original image and calculate the Distance Transform (DT) [15] [16] of the image, that is formed by the detected edges. The distance between the edge measurements and the detected edges is calculated using the image produced from DT. Finally, by using the Gauss-Newton algorithm, we minimize the distance between the edge measurements of the virtual model and the detected edges from the real object.

 ISBN: 978-1-61208-724-5

Section II provides a formulation of the 3D visual tracking problem. In Section III we present the architecture and of our visual tracking system and the experimental results on simulated and real data are shown in Section IV. Finally, in Section V we discuss about the merits of our method and provide some cues for further improvement.

II. FORMULATION

We treat the 3D visual tracking problem, as a procedure of estimating the camera pose (3D position and 3D orientation of the camera) relative to the object, i.e., estimate the extrinsic parameters of the real camera that led to the projection of the object onto the image. With the use of a known model of the object, we construct a virtual camera that projects and imposes the model to the real image. By finding the distance between the projection of the virtual object (model) and the real object (object in the real scene, that we want to track), we are able to estimate the extrinsic parameters of the virtual camera that minimize this distance, with the use of the Gauss-Newton iterative algorithm. If the distance of the projection of the two objects is nearly zero (global minimum), the extrinsic parameters of both cameras match, and the pose of the real camera is adequately estimated.

A. Camera Model

For our purposes, we use the full camera model, which is described by the projection matrix P, that projects each world point of a 3D scene to the image plane of the camera. The projection matrix is constructed by the multiplication of the intrinsic and extrinsic parameters matrices of the camera. We use the following intrinsics matrix, which maps a 3D point in camera coordinates (x_c, y_c, z_c) to a 2D point in pixel coordinates (u, v),

$$\boldsymbol{K} = \begin{bmatrix} f_p & 0 & u_0 & 0 \\ 0 & f_p & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{3\times 4} \tag{1}$$

where f_p is the focal length in pixel units and (u_0, v_0) is the principal point i.e., the point of intersection between the camera's optical axis and the image plane. We do not take into account any lens distortion that may occur by defects in lens design and manufacturing or by the nature of the lens. In this approach the intrinsic parameters are known and fixed, so the camera we use is calibrated. In a future implementation of our algorithm as a mobile phone application, camera calibration could easily be incorporated during installation or first use on the end-user's phone. During this stage, the end-user can be guided through the process of calibration, following the procedure explained in [17].

The extrinsic parameters matrix holds the position and orientation of the camera in world coordinates and it maps a 3D point in world coordinates to a 3D point in camera coordinates. This matrix is defined as follows,

$$\boldsymbol{E} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^\top & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \tag{2}$$

where $\boldsymbol{R} = \boldsymbol{R}(\theta_y)\boldsymbol{R}(\theta_z)\boldsymbol{R}(\theta_x) \in \mathbb{R}^{3\times 3}$ is a 3D rotation matrix represented in Euler angles and $\boldsymbol{t} = [t_x, t_y, t_z]^\top \in \mathbb{R}^3$ is a position vector. The matrix in (2) describes the position and orientation of the object relative to the observer - camera. In our approach, we want to estimate the camera pose, i.e., the position and orientation of the observer - camera relative to the object. To achieve this, we calculate the inverse of the aforementioned matrix as,

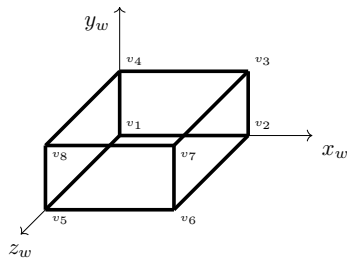


Figure 1. Model in world coordinates.

$$\boldsymbol{E}_c = \begin{bmatrix} \boldsymbol{R}^\top & -\boldsymbol{R}^\top \boldsymbol{t} \\ \boldsymbol{0}^\top & 1 \end{bmatrix} \tag{3}$$

B. Cuboid Object

We are using a cuboid object (box) for tracking, which is depicted in Figure 1. The model is composed of a set of eight vertices $\mathcal{V} = \{\boldsymbol{v} : \boldsymbol{v} \in \mathbb{R}^3\}, |\mathcal{V}| = 8$ and a set of six surfaces $\mathcal{S} = \{\boldsymbol{s} : \boldsymbol{s} \subset \mathcal{V}\}, |\mathcal{S}| = 6, |\boldsymbol{s}_i| = 4, 1 \leq i \leq |\mathcal{S}|$. Because there are no other objects in the scene, the model's local coordinates are equal to the scene's global coordinates. With the use of the known intrinsic parameters $\boldsymbol{K}$ and an initial hand-picked estimation of the real camera's extrinsic parameters $\boldsymbol{E}^{(0)}$, we construct a virtual camera, that is being defined by the following projection matrix,

$$\begin{aligned} \boldsymbol{P} &= \boldsymbol{K}(f_p, u_0, v_0)\boldsymbol{E}_c^{(0)}(t_x, t_y, t_z, \theta_x, \theta_y, \theta_z) \\ &= \boldsymbol{K}(f_p, u_0, v_0)\boldsymbol{E}_c(\boldsymbol{x}^{(0)}) \end{aligned} \tag{4}$$

where $\boldsymbol{x}^{(t)}$ is a vector containing the extrinsic parameters at time t.

Each vertex $\boldsymbol{v}_i$ of the cuboid is projected to the image plane of the virtual camera as,

$$\tilde{\boldsymbol{p}}_{\boldsymbol{i}} = \boldsymbol{P}\tilde{\boldsymbol{v}}_{\boldsymbol{i}} \tag{5}$$

where $\tilde{\boldsymbol{p}}_i = [x_p^{(i)}, y_p^{(i)}, z_p^{(i)}]^\top$ and $\tilde{\boldsymbol{v}}_i = [x_w^{(i)}, y_w^{(i)}, z_w^{(i)}, 1]^\top$ are expressed in pixel and world homogeneous coordinates, respectively. The conversion to cartesian pixel coordinates is done as follows,

$$\boldsymbol{p}_i = \begin{bmatrix} \frac{x_p^{(i)}}{z_p^{(i)}} & , & \frac{y_p^{(i)}}{z_p^{(i)}} \end{bmatrix}^\top = \begin{bmatrix} u_i \\ v_i \end{bmatrix} \tag{6}$$

C. Parameter Estimation

As we have mentioned in Section II-A, we model our camera in such a way, that the intrinsic parameters are fixed. In this case, the camera is said to be calibrated i.e., we have a prior knowledge of the camera's intrinsic parameters. This reduces the degrees of freedom (DoFs) for parameter estimation to only six; the extrinsic parameters of the camera (three DoFs for camera position and three DoFs for camera orientation).

The extrinsic parameters $\hat{\boldsymbol{x}}$ can be estimated as the minimization of the squared sum of the reprojection errors between $\boldsymbol{v}_i$ and $\boldsymbol{p}_i$,

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \sum_i \parallel \boldsymbol{P}(\boldsymbol{x})\tilde{\boldsymbol{v}}_i - \boldsymbol{p}_i \parallel^2 \tag{7}$$

 ISBN: 978-1-61208-724-5

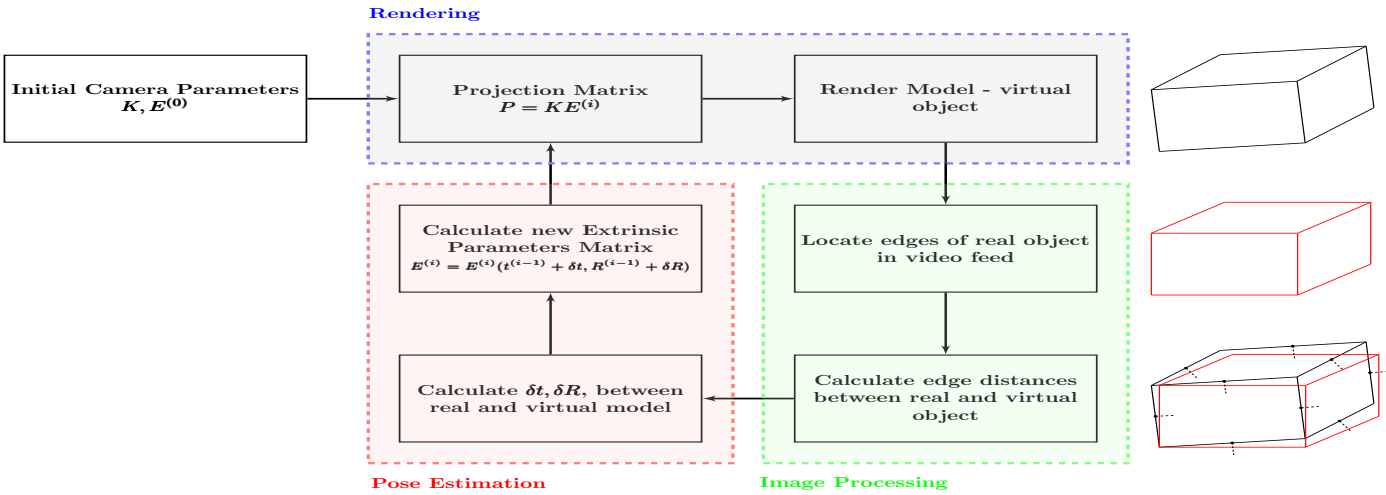


Figure 2. Visual Tracker work-flow, illustration of the main building blocks a) Rendering (blue), b) Image Processing (green) and c) Pose Estimation (red).

where $\boldsymbol{P}$ and $\boldsymbol{x}$ are the projection matrix and the extrinsic parameters, as shown in (4). The pose estimation problem can be solved by minimising the sum of residual errors, $r_i = \| \boldsymbol{P}(\boldsymbol{x})\tilde{\boldsymbol{v}}_i - \boldsymbol{p}_i \|$. Equation (7) can also be written in vector form as,

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \| f(\boldsymbol{x}) - \boldsymbol{b} \|^2 \tag{8}$$

where again $\boldsymbol{x}$ is a vector that contains the extrinsic parameters, $\boldsymbol{b}$ is vector containing some type of measurements made on the image (detected features) and $f(\cdot)$ is a function that relates vectors $\boldsymbol{x}$ and $\boldsymbol{b}$ (projection matrix as seen in (4)).

Function $f(\cdot)$ is usually of a non-linear nature, due to the perspective projection transformation. So in our case we use the Gauss-Newton optimization algorithm, which is a non-linear least squares technique [18]. With the use of prior knowledge of an initial state $\boldsymbol{x}_0$, the residual error between consecutive states within a time frame τ, is minimized as

$$\begin{aligned} \boldsymbol{x}_{k+1}^{(\tau)} &= \boldsymbol{x}_k^{(\tau)} + \delta_{\boldsymbol{x}} \\ &= \boldsymbol{x}_k^{(\tau)} - \boldsymbol{J}_k^{\dagger(\tau)} \boldsymbol{\epsilon}_k^{(\tau)} \end{aligned} \tag{9}$$

where $\delta_{\boldsymbol{x}} = -\boldsymbol{J}_k^{\dagger(\tau)} \boldsymbol{\epsilon}_k^{(\tau)}$ is the minimization step defined by the Gauss-Newton algorithm, $\boldsymbol{J}_k^{\dagger(\tau)}$ is the pseudo-inverse of $\boldsymbol{J}_k^{(\tau)}$, the Jacobian of $f(\boldsymbol{x}_k^{(\tau)})$ and $\boldsymbol{\epsilon}_k^{(\tau)} = f(\boldsymbol{x}_k^{(\tau)}) - \boldsymbol{b}^{(\tau)}$ is the residual error at iteration k.

III. Visual Tracking System

Our method consists of three main building blocks: The first building block, Rendering, is responsible for the correct projection of the known virtual 3D object to the image plane. The second building block, Image Processing, consists of image processing methods for scanning every frame of the video feed from the camera and identifying 2D image features, which are likely to describe the object of the scene. In addition, it measures the distance between the projection of the virtual object and the extracted image features. The last building block, Pose Estimation, uses a non linear fitting method (Gauss-Newton), for accurate estimation of the 3D position and orientation of the camera, so that the projection of the virtual object matches the projection of the real object. Figure 2 illustrates the general procedure of our 3D visual tracker, which consists of the aforementioned building blocks.

A. Rendering

The rendering procedure is responsible for rendering the known model of the object we would like to track. By using the virtual model as shown in Figure 1, we project each of its vertices $\boldsymbol{v}_i$ to the image plane (5). The projection matrix $\boldsymbol{P}$ is constructed as shown in (4), which models our virtual camera. Each vertex is projected to a 2D point $\boldsymbol{p}_i$ expressed in pixel coordinates. Subsequently, visibility culling techniques are applied to the projected object, to determine if it is visible form the virtual camera.

The first technique is called *front camera visibility*, where for each vertex $\boldsymbol{v}_i$ we calculate its distance d relative to the camera position $\boldsymbol{t}$. If any distance $\boldsymbol{d}$ is smaller than f_m (camera's focal length in metric units), then this vertex is not visible and in this case, we cull the whole object. Secondly, *back face culling* is used to determine which surfaces of the model are facing the camera. For each surface, the angle θ is calculated between the camera's look vector, from the camera's position $\boldsymbol{t}$ and a vertex of the surface, and the normal of the surface. If the angle is smaller than $90°$, then the surface is facing the camera and it should be rendered. Finally, *edge clipping* is applied, where each edge of the imposed model, is clipped along the borders of the image.

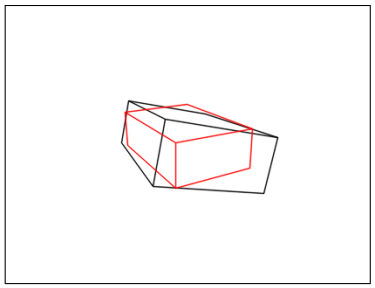
The output of the rendering procedure, for simulated and real data, are illustrated in Figure 3.

B. Image Processing

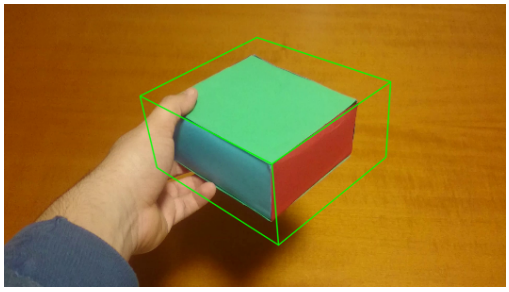
The second main building block of our visual tracking system, Image Processing, is responsible for dividing each projected edge of the virtual models into *control points* and calculating the minimum distances between them and the image features extracted from the real image. These control points help in measuring the spatial difference between the virtual and real projected object and give an estimate of how much the extrinsic parameters of the virtual camera have to be altered, so the edges of the virtual object match the edges of its real counterpart.

For each visible projected edge of the virtual model, we form its direction vector as $\boldsymbol{o}_i = \boldsymbol{p}_i^{(2)} - \boldsymbol{p}_i^{(1)}, i = 1, \cdots, |\mathcal{N}|$, where $\mathcal{N}$ is the set of all the visible projected edges, $|\mathcal{N}| = N$ is their corresponding number and $\boldsymbol{p}_i^{(1)}, \boldsymbol{p}_i^{(2)}$ are the projected

 ISBN: 978-1-61208-724-5

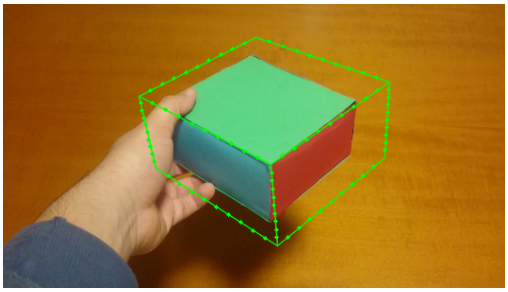


(a)

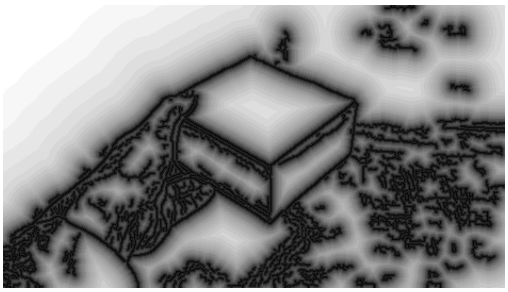


(b)

Figure 3. (a) The red and black cuboid represent our simulated data and the virtual model, respectively, (b) A frame from a real video-feed and the imposed virtual model (green cuboid).



(a)



(b)

Figure 4. (a) Control points for each edge of virtual object, (b) distance transform.

vertices of the cuboid and the endpoints of the i-th edge. The control points are then calculated as,

$$\boldsymbol{m}_{ij} = \boldsymbol{p}_i^{(1)} + j\frac{\boldsymbol{o}_i}{M-1} \tag{10}$$

where $\boldsymbol{m}_{ij}$ is the j-th control point of the i-th edge and M are the number of control points for each edge. We omit control points at the endpoints of the edge so $j = 1, \cdots, M-1$ and the correct number of control points are $M-2$. Figure 4a illustrates the calculated control points for the real data scenario.

The next step of the image processing procedure is to extract features from the real image. The type of features that we are detecting are edges, which appear as intensity discontinuities on the image. For the purposes of our method, we used the Canny edge detection algorithm [19]. Because the real image of the simulated data is rendered by us, the only information that appears on it are the projected edges so there is no need for edge detection. This is not the case for real data, as every frame of the image contains a lot of information with a high quantity of noise. The use of Canny edge detection algorithm in this case is mandatory. We also want to remove as much noise we can, so the two thresholds for the hysteresis procedure of the algorithm, are set to high values, to make sure that edges that represent noise or small and weak edges of the scene are discarded.

The final step of image processing is about finding and measuring the distances between the control points and their closest image features, i.e. the edges that have been extracted in the previous step. To achieve this, we apply the Distance Transform to the image being produced by the Canny algorithm. This results to a new image, where the value of each point is the Euclidean distance between that point and its nearest image feature, as seen in Figure 4b.

Formalizing distance transform in the context of our method, we can write,

$$d_{ij} = DT(\boldsymbol{\mathcal{F}})[\boldsymbol{m}_{ij}] \equiv \min_{\boldsymbol{f}} dist(\boldsymbol{m}_{ij}, \boldsymbol{f}) \tag{11}$$

where d_{ij} is the distance measurement for each $\boldsymbol{m}_{ij}$ (control point), $\boldsymbol{\mathcal{F}} = \{\boldsymbol{f}_1, \cdots, \boldsymbol{f}_E\}, \boldsymbol{f}_e \in \Re^2$ is the set of all edge pixels $\boldsymbol{f}_e$ that have been extracted, $|\mathcal{F}| = E$ is the number of all edge pixels and $dist(\boldsymbol{m}_{ij}, \boldsymbol{f}) = \| \boldsymbol{m}_{ij} - \boldsymbol{f}_e \|_2$ is the Euclidean distance between $\boldsymbol{m}_{ij}$ and $\boldsymbol{f}_e$.

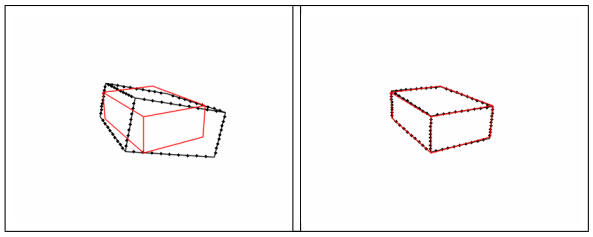
The distance d_{ij} for each control point $\boldsymbol{m}_{ij}$ can be found by treating the distance transform image as a lookup table, i.e., just lookup the value from the pixel of the distance transform image with the same position as the $\boldsymbol{m}_{ij}$ pixel position.

The main advantage of using DT is that is easy to express the distance of each edge pixel $\boldsymbol{f}_e$ and control point $\boldsymbol{m}_{ij}$ in closed form w.r.t. extrinsic parameters of the camera. So using this in the context of the Gauss-Newton algorithm, as explained in Section III-C, is straightforward.

C. Pose Estimation

The last building block, Pose Estimation, is the main procedure of the 3D visual tracker. In this phase, based on the measurements that have been made to the real image, we try to minimize the difference between the extrinsic parameters of the virtual and real camera. To achieve this, we use the Gauss-Newton algorithm, as it has been explained in Section II-C. In our case, the state vector $\boldsymbol{x} = [x_1, x_2, x_3, x_4, x_5, x_6] \equiv [t_x, t_y, t_z, \theta_x, \theta_y, \theta_z]$, contains the extrinsic parameters of the virtual camera and $f(\boldsymbol{x})$ is the function that maps the distances that are found for the control points $\boldsymbol{m}_{ij}$, w.r.t. the extrinsic

 ISBN: 978-1-61208-724-5



(a) (b)

Figure 5. Fitting of model cube (black) to real cube (red), after 5 iterations.

parameters. The residual error ϵ equals to,

$$\boldsymbol{\epsilon}_k = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{NM} \end{bmatrix} = \boldsymbol{d}_k = \begin{bmatrix} d_{1,1} \\ d_{1,2} \\ \vdots \\ d_{N,M} \end{bmatrix} \tag{12}$$

where $\boldsymbol{d}_k$ is a vector that contains the distances for all control points, for the k-st iteration of the Gauss-Newton algorithm. To iteratively minimise the residual errors we use (9). Here, the objective function that we are trying to minimize is $\boldsymbol{f}(\boldsymbol{x}) = \sum_{i=1}^{NM} r_i(\boldsymbol{x}) = \sum_{i=1}^{N} \sum_{j=1}^{M-1} d_{ij}(\boldsymbol{x})$, where $r_i(\boldsymbol{x}) = d_{ij}(\boldsymbol{x})$ is the Euclidean distance between control point $\boldsymbol{m}_{ij}$ and edge pixel $\boldsymbol{f}_k$, as a function of $\boldsymbol{x}$. Therefore, the Jacobian of $\boldsymbol{f}(\boldsymbol{x})$ for iteration k equals to,

$$\boldsymbol{J}^{(k)} = \nabla f(\boldsymbol{x}) = \left[\frac{\partial \boldsymbol{d}(\boldsymbol{x})}{\partial x_1}, \cdots, \frac{\partial \boldsymbol{d}(\boldsymbol{x})}{\partial x_S} \right] \in \Re^{MN \times 6} \tag{13}$$

where x_s is one of the 6 parameters of the state vector $\boldsymbol{x}$ and $\boldsymbol{d}(\boldsymbol{x})$ is the function that maps the measured distances from the distance transform, with the extrinsic parameters, i.e., the pose of the camera.

By minimising this quantity, the distance between the control points and image features will also be minimized. At this point, the real object's edges match the virtual object's edges, in which case the pose parameters can be inferred.

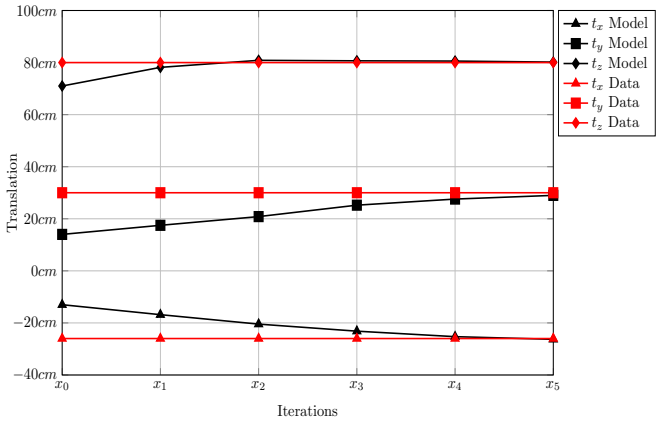
IV. Results

In this section, we present some experiments that we have made with our visual tracker both on simulated and real data, so we can determine it's strengths and weaknesses.

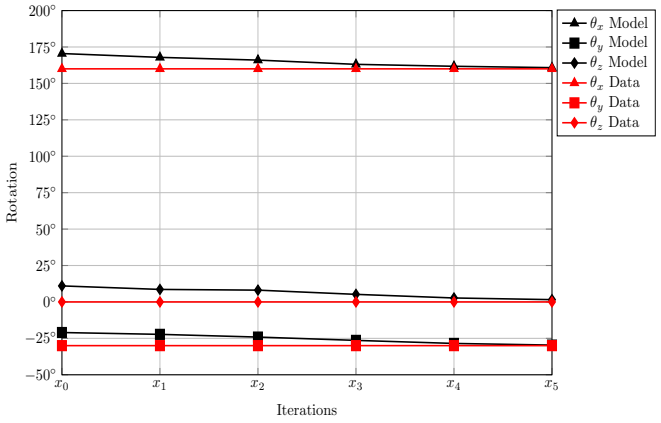
A. Simulated Data Experiments

For our experiments on simulated data, we have rendered another cuboid with the use of a computer generated (CG) camera. The intrinsic parameters for both cameras are set to the same values. The CG camera extrinsic parameters are set to $\boldsymbol{x}_{\text{CG}} = [t_x, t_y, t_z, \theta_x, \theta_y, \theta_z] = [-26\text{cm}, 30\text{cm}, 80\text{cm}, 160°, -30°, 0°]$ and we use them as ground truth. The virtual camera extrinsic parameters are set to $\boldsymbol{x}_{\text{virtual}} = [-13\text{cm}, 14\text{cm}, 71\text{cm}, 170.5°, -21°, 11°]$. Finally, we render on the same image, using both cameras, the cuboid depicted on Figure 1.

As we can see from Figure 5, after 5 iterations, the pose of the virtual object matches the simulated object's pose. The final virtual camera extrinsic parameters we get after the end of the pose estimation procedure are $\boldsymbol{x}_{\text{virtual}} = [-26.3cm, 29cm, 80.2cm, 160.8°, -29.6°, 1.6°]$, which are really close to the extrinsic parameters of the CG camera. As



(a)



(b)

Figure 6. Convergence between virtual and simulated camera extrinsic parameters, a) cameras position, b) cameras orientation.

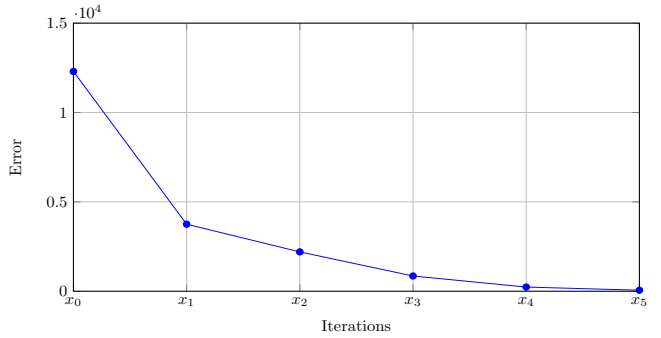
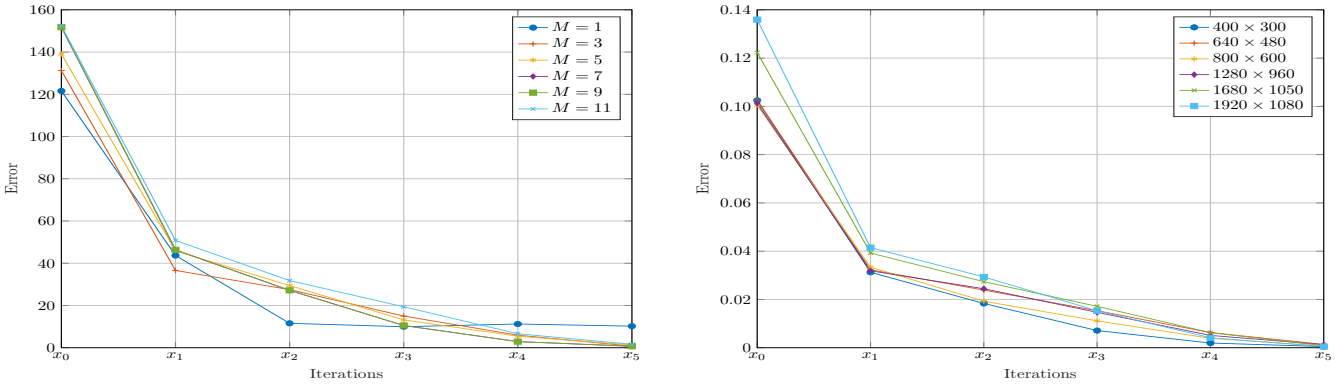


Figure 7. Error between simulated and virtual cuboid for each iteration.

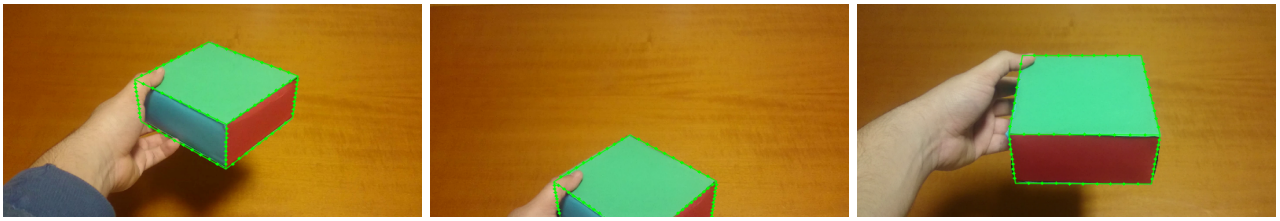
shown in Figure 6, with each iteration k, each parameter of the state vector $\boldsymbol{x}_{\text{virtual}}$ gradually converges to the the ground truth parameters. We can actually notice the gradient descent step, as for each time we get closer to the minimum, the next step tends to be smaller, because the gradient magnitude decreases.

The error between the two cuboids is calculated as the squared sum of the distances d, $E_k = \boldsymbol{d}_k \cdot \boldsymbol{d}_k = \sum_{i=1}^{MN} d_i^{2\ (k)}$,



(a) (b)

Figure 8. Error evaluation a) for various number of control points, b) for various image resolutions



(a) (b) (c)

Figure 9. a) Fitting of virtual cube on a real data image, b) Edge clipping along the borders of the image, c) Fitting of virtual cube with two surfaces visible.

for each iteration k, which is the square of the objective function $f(\boldsymbol{x})$ calculated at the extrinsic parameters $\boldsymbol{x}_k$. From Figure 7, it is clear that the error drops exponentially, another clue that the pose estimation was correct. Again, the exponential rate of the error has the same traits as the convergence of the extrinsic parameters.

As seen in Figure 8a, in all cases error converges to 0, except the case where $M = 1$ because a small amount of measurements are extracted from the frame. Furthermore, if $M \geq 11$ despite the fact that the error after 5 iterations is close to 0, the time that is needed for each frame for the fitting procedure starts to exceed the acceptable limits ($33ms$ time cap). This is even worse in the case where we are using real data, as the number of iterations is doubled to 10. Here the error is normalized w.r.t. number of the control points for all visible edges $\hat{E}_k = E_k/MN$. Also in Figure 8b , we show that our tracking method works well for various number of resolutions and aspect ratios. In all cases, after 5 iterations the squared distance between the virtual and the simulated object converges to 0. Here the error is normalized with the image area $\hat{E}_k = E_k/(Width \times Height)$.

With frames of 400×300 we ensure that our data contain sufficient information for our models to extract the correct pose, while not posing a significant computational overhead. Likewise, we choose to operate with $M = 9$ in an effort to ensure that a good majority of the elements in $\boldsymbol{d}_k$ of (12) indeed holds the distances between truly corresponding points in the model and data.

B. Real data experiments

For our real data experiments, we have used a box of cuboid shape. In this scenario, we altered the intrinsic parameters of the virtual camera, so they match the intrinsics of the real camera. The fitting of the cuboid in Figure 9a is achieved after the 3 first frames of the video. This happens because, even after we doubled the number for iterations for each frame ($k = 10$), the image gradient of the distance transform for each frame is calculated with the use of the Sobel operator, which smooths the produced image. This smoothing results to smaller Δ steps at each iteration. Nevertheless, we want to keep it that way, so it does not affect the overall accuracy of our visual tracker and the first 3 frames work as an initialization stage.

Further down the same video feed, the real object is clipped along the four edges of the image plane, in Figure 9b. Even at these conditions our tracker manages to successfully track the real object. Of course, there is a limit to the portion of the object that is being clipped, which is about half of the object. After that point, the pose estimation of the tracker becomes unstable and at some point it completely loses the correct pose of the real camera. In general, when we want to track some object on a real scene, we usually set the pose of the real camera to be looking straight at the object and the object being in the middle of the image plane. So, conditions where half or more of the object is clipped along some edge of the image, are not expected in real applications, such as Augmented Reality.

The final experiment was conducted in order to determine how many surfaces of the cuboid should be visible, for the visual tracker to correctly estimate the camera's pose. We have concluded that at least two surfaces should be always visible so the estimation of the state vector $\boldsymbol{x}$ is stable. In fact, at least

 ISBN: 978-1-61208-724-5

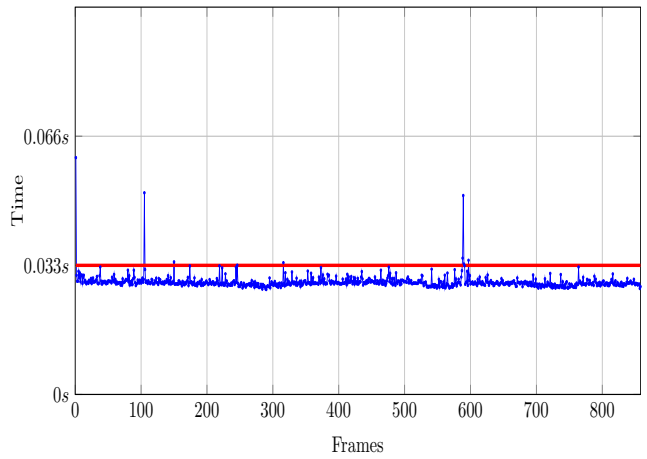


Figure 10. Elapsed time for the visual tracker process on each frame. Red line is the 33 milliseconds cap, for real time performance.

6 rows of $\boldsymbol{J}$ have to be linearly independent from each other, to be possible to calculate correctly the pseudo-inverse of $\boldsymbol{J}$. The $\boldsymbol{m}_{ij}$'s along each edge are linearly dependable, so we need at least 6 visible edges to surpass the previous constraint. For at least 6 edges to be visible, we need at least 2 surfaces of the object to be visible. This is the case in Figure 9c, where the cuboid is correctly tracked.

Finally, our visual tracker can run in real-time, i.e., process each frame of the video feed and estimate the camera's pose in under 33 milliseconds, which is the time interval between two consecutive frames in a video feed of 30 frames per second, as shown in Figure 10. This is due to the fact that, even if three surfaces of the cuboid are visible, we have to render 9 edges of the model. Each edge contains $M = 9$ control points and the overall number of control points are 81. So, for each frame we have to compute the camera's pose, using only 81 distances for the non-linear fitting iterations, which is a small number of data to process for modern workstations. This leads to the fact that we can impose correctly a virtual object that matches the real one, without adding any extra delay to the video's sequence. The experiments were conducted on a 3.1GHz dual-core processor with 8GB of RAM.

V. Conclusion and Future Work

In this paper, we have presented an approach to model-based 3D visual tracking with the use of Distance Transform, which gives us an estimation of how far or close the virtual projected object is, relative to the real object we want to track. With the use of this estimation, we were able to measure the difference between the extrinsic parameters of the virtual and the real camera. Then we proceeded to the minimization of this difference using the Gauss-Newton algorithm, which led to the estimation of the real camera's pose in real-time, for each frame of a video feed.

One main problem of our method is that it does not handle very well cases when the object we want to track is partially occluded. To overcome this problem, we can explore techniques that find outlier measurements from the image, like RANSAC [10], and thus be able to remove features that are being extracted and that belong to the occluding object. Another approach is to find correspondences between the 2D (non-occluded) projections of the real object and the 2D or 3D control points of the virtual object, so the pose estimation process is based solely on them.

The initial extrinsic parameters of the camera are hand picked, so that the projected virtual object bounds the real one. An investigation on automatic initialization of the initial parameters will also be conducted. Finally, the renderer of our implementation needs to be expanded, so it can handle more complex objects, non-convex objects. To achieve this, the rendering procedure needs to support *z-buffering* techniques.

Although we use a simplistic model (cuboid) for our tracking experiments, our method can be expanded to more complex and non-convex objects, according to the targeted application.

With the aforementioned improvements implemented, it will be possible to compare our method with techniques like [13] or [14]. To the best of our knowledge, there is not any public source code for these two techniques, so their implementation is left for future work.

References

[1] J. Barandiaran and D. Borro, "Edge-based markerless 3d tracking of rigid objects," in 17th International Conference on Artificial Reality and Telexistence (ICAT 2007), Esbjerg, Jylland, Denmark, Nov 2007, pp. 282–283.

[2] M. L. et al., "Fast object localization and pose estimation in heavy clutter for robotic bin picking. the international journal of robotics research, 31(8), 951-973," International Journal of Robotic Research - IJRR, vol. 31, 07 2012, pp. 951–973.

[3] D. Berenson and S. S. Srinivasa, "Grasp synthesis in cluttered environments for dexterous hands," in Humanoids 2008 - 8th IEEE-RAS International Conference on Humanoid Robots, Daejeon, South Korea, Dec 2008, pp. 189–196.

[4] E. Marchand, H. Uchiyama, and F. Spindler, "Pose estimation for augmented reality: A hands-on survey," IEEE Transactions on Visualization and Computer Graphics, vol. 22, 01 2016, pp. 2633–2651.

[5] V. Lepetit and P. Fua, "Monocular model-based 3d tracking of rigid objects," Found. Trends. Comput. Graph. Vis., vol. 1, no. 1, Jan. 2005, pp. 1–89. [Online]. Available: http://dx.doi.org/10.1561/0600000001

[6] W. A. Hoff, K. Nguyen, and T. Lyon, "Computer vision-based registration techniques for augmented reality," in Intelligent Robots and Computer Vision XV, 1996, pp. 538–548.

[7] D. K. et al., "Real-time vision-based camera tracking for augmented reality applications," in Proceedings of the ACM Symposium on Virtual Reality Software and Technology, ser. VRST '97. Lausanne, Switzerland: ACM, 1997, pp. 87–94. [Online]. Available: http://doi.acm.org/10.1145/261135.261152

[8] I. Skrypnyk and D. G. Lowe, "Scene modelling, recognition and tracking with invariant image features," in Third IEEE and ACM International Symposium on Mixed and Augmented Reality, Arlington, VA, USA, Nov 2004, pp. 110–119.

[9] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, vol. 60, no. 2, Nov 2004, pp. 91–110. [Online]. Available: https://doi.org/10.1023/B:VISI.0000029664.99615.94

[10] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," Commun. ACM, vol. 24, no. 6, Jun. 1981, pp. 381–395. [Online]. Available: http://doi.acm.org/10.1145/358669.358692

[11] W. L. et al., "SSD: Single shot multibox detector," in European Conference on Computer Vision (ECCV), vol. 9905, Amsterdam, The Netherlands, 10 2016, pp. 21–37.

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, June 2016, pp. 779–788.

 ISBN: 978-1-61208-724-5

[13] C. Harris and C. Stennett, “RAPiD - a video rate object tracker,” in In Proc. British Machine Vision Conference BMVC ’90, Oxford, UK, 1990, pp. 73–78.

[14] T. Drummond and R. Cipolla, “Real-time tracking of complex structures with on-line camera calibration,” in In Proc. British Machine Vision Conference BMVC ’99, Nottingham, UK, 1999, pp. 574–583.

[15] A. Fitzgibbon, “Robust registration of 2d and 3d point sets,” Image and Vision Computing, vol. 21, 04 2002, pp. 1145–1153.

[16] D. Huttenlocher, “Cs664 computer vision - 7. distance transforms,” 2008, [retrieved: May, 2019]. [Online]. Available: https://www.cs.cornell.edu/courses/cs664/2008sp/handouts/cs664-7-dtrans.pdf

[17] Z. Zhang, “A flexible new technique for camera calibration,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 11, Nov 2000, pp. 1330–1334.

[18] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, “Bundle adjustment - a modern synthesis,” in Proceedings of the International Workshop on Vision Algorithms: Theory and Practice, ser. ICCV ’99. London, UK, UK: Springer-Verlag, 2000, pp. 298–372. [Online]. Available: http://dl.acm.org/citation.cfm?id=646271.685629

[19] J. Canny, “A computational approach to edge detection,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, Nov 1986, pp. 679–698.

 ISBN: 978-1-61208-724-5