Contents

List of Figures

CHAPTER 1

Introduction

Head tracking is a very important and challenging topic in the field of computer vision. The possibility to track a human head throughout long video sequences and to recover the parameter vector ϕ which describes the pose of the head in the space has a great impact in fields like face recognition, expression interpretation and image coding. All these problems are more likely to be solved robustly if it is possible to retrieve a stabilized image either of the head or of the face. Moreover, determining the trajectory of the head in the 3D space plays a fundamental role both in the development of vision driven human-computer interfaces (HCI) and gesture recognition systems as well as in augmented reality applications.

1.1. Head Tracking Systems Architecture Overview

Many of the head tracking algorithms which estimate the pose of the head in the 3D space try to accomplish this task by applying an iterative updating procedure on the parameter vector that represents the pose of the head at each frame. Usually this parameter vector has six components: three of them (angle components) are used to represent the orientation of the head and the others three to represent the position of the head (translation components).

The basic idea is to consider some quantity related directly to the pose of the head (let this quantity be $V(\phi(t))$), whose value can be also recovered from some computation carried out on the video stream (in this case we address this value with $V(I(t))$, where $I(t)$ represents the image on the image plane at time t). Introducing a suitable norm we can define the cost functional:

$$J(\phi(t)) = \|V(\phi(t)) - V(I(t))\| \tag{1.1.1}$$

so that the estimation of the parameter vector that describe the pose of the head is reduced to the following optimization problem:

$$\phi^*(t) = arg \min_{\phi(t)\in\Phi} J(\phi(t))$$

As far as the quantity V is concerned, there are many possible choices: V may represent the position on the image plane of some head feature (in this case its value can be recovered both using some template matching procedure applied

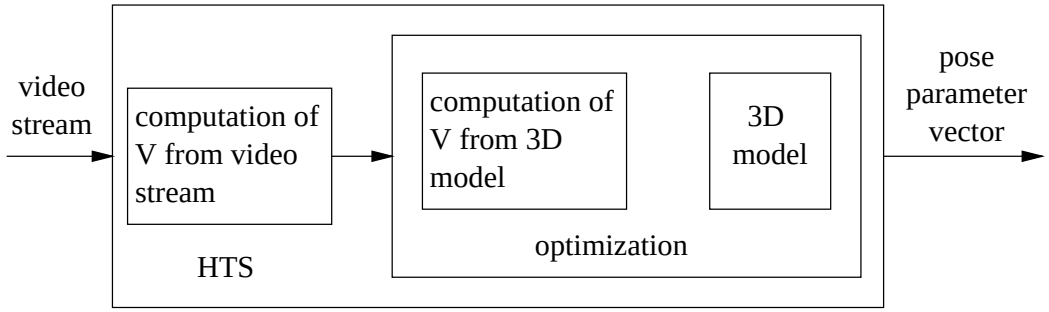

FIGURE 1.1.1. General architecture of an head tracking system.

to the image $I(t)$ and back projecting the corresponding point on the internal 3D model on the image plane), the components of the optical flow (also in this case computable both from the video stream and from the trajectory of the 3D model), and so on so forth.

According to this general framework (see figure (1.1.1)), once the structure of V has been defined, an head tracking algorithm is basically characterized by three aspects:

(1) The procedure to compute V from the video stream.
(2) The procedure to compute V from the a suitable internal model.
(3) The optimization method.

Each of the previous choices play a crucial role in determining which will be the performances, the robustness and generality of the algorithm.

1.2. Tracking Algorithm Requirements

It is hard to realize a good head tracking system (HTS): difficulties arise from the complex head geometry, from the large and rapid motions and deformations that may happen during the tracking and also from the fact that the scene in which the head is inserted may have a background that may distract the tracking algorithm. A good HTS should satisfy the following prerequisites:

- It should be *markerless*. The head tracking system should not rely on any marker attached to the human head and moreover it should not require any of the head features (such for example eyes, nose, mouth, ears) to be visible over the entire video sequence.
- It should be *accurate*. This means that the system should be able to track and calculate the pose of the head over long time periods, using just one single and uncalibrated camera and possibly without introducing any kind of cumulative error during the tracking, given *any* arbitrary complex trajectory of the head.

- It should be *robust*. The system should not be affected by varying light conditions, local head deformations (due for example to changes of expression) and should also be able to handle partial head occlusions. Moreover its performance should not depend heavily on some set of parameters that need to be tuned properly for each different sequence.
- It should be *fast*. Many of the situations where an head tracking system may be used require the algorithm to run in real time; thus the procedure should be fast enough to be implemented real time without using expensive hardware.
- It should be *easy to initialize*. The first fundamental task that needs to be accomplished in order to track the head is to find the head on the initial frame and to provide the algorithm with an initial estimate of the pose of the head. Both these two task need to be carried out automatically, without affecting the performance of the algorithm even though the initial pose estimation is not so precise.

1.3. Related Works

In this section we will present a brief description of some of the most interesting contribution to the head tracking problem, trying to study these algorithms according to the framework that has been described in section (1.1).

1.3.1. Basu, Essa and Pentland's approach [10]. The method described by the authors to recover the head trajectory is based on the matching of the optical flow, which is calculated both directly from the video stream and using a 3D ellipsoidal model of the head. According to the notation used in section (1.1), we have that the function V, which is defined for each point belonging the portion of the image $I(t)$ that represents the head, can be expressed as:

$$V(x, y, t) = \begin{pmatrix} u(x, y, t) \\ v(x, y, t) \end{pmatrix}$$

where u and v are the components of the optical flow at time t for the pixel at the position[1] $\begin{pmatrix} x & y \end{pmatrix}^T$. The basic idea of the algorithm is to compute the unconstrained optical flow for the whole video sequence and then the rigid motion of the 3D head model that best fits the observed flow is interpreted as the head trajectory. The optimization procedure is carried out using the simplex gradient descent technique. The authors claim that this method is stable over extended sequences and not too sensitive to the initial fit of the 3D model: the results of

[1]Note that when the quantity $V(x, y, t)$ is calculated using the 3D model, the position $\begin{pmatrix} x & y \end{pmatrix}^T$ is obtained by back projecting on the image plane the corresponding point on the 3D model.

the tracking for a synthetic sequence (for which the ground truth values of the pose parameters are known) show a maximum error for the Euler angles of 14°. Unfortunately, as far as the translation is concerned, the values are given in pixels, and this fact does not allow a precise quantitative evaluation of the performance of the algorithm[2]. Another important issue is related to the computation of the optical flow, that is carried out at the beginning of the tracking procedure for the *entire* video sequence: since the optical flow computation is in general a time consuming activity, this fact may seriously affect the possibility to implement such algorithm in real time. Finally no tests under varying light conditions are performed: we believe that shadows cast on the head may seriously affect the method, since dark shadow regions are poor of texture and this may severely affect the optical flow retrieval.

1.3.2. La Cascia, Scarloff and Athitsos's approach [11]. The authors focus their attention in the development of an HTS able to track robustly the head also under varying light conditions. The internal head model is a texture mapped cylinder and the matching procedure is intended to fit the region of the image corresponding to the head with the back projection of the texture present on the internal model. To cope with varying light conditions, the authors use an illumination-adjusted version of the texture present on the 3D model. According to the notation used in section (1.1), in this case the function V is a scalar function:

$$V(x, y, t) = l$$

where l is the adjusted luminance value of the pixel $\begin{pmatrix} x & y \end{pmatrix}^T$.

Thanks to an extensive set of experiments where the output of the algorithm is compared with the ground truth values recovered using a "flock of bird" magnetic sensor attached to the head, the authors claim that the algorithm is both able to track the head robustly also under varying light conditions. The algorithm is said to be quite insensitive to perturbations of the initial positioning of the 3D model. In their experiments the precision of the tracker is defined as the root mean square error computed over the sequence up to the point where the track is lost:

$$E_i^2 = \frac{1}{N} \sum_{j=1}^{N} e_i^2(j)$$

The quantity $e_i(j)$ is the Mahalanobis distance between the estimated and measured value for the parameter ϕ_i at time j. The authors define the track to be

[2]One meaningful comparison is obtained comparing the error of the estimation of the translation with the head dimensions: this will allow us to decide whether the error is "great" or acceptable.

lost when $e_i(j) \geq 2$. Unfortunately it is not possible to associate to the threshold value for $e_i(j)$ a clear physical meaning and from the results described in the paper it is also impossible to determine which is the maximum tracking error for each component of the pose vector. Anyway from a visual inspection of the graphs present in the paper the worst case error for the translation parameters seem to be about $7.5cm$ and for the rotation parameters about $10°$.

1.3.3. Preteux and Malciu's approach [12]. The principle of this method is based on the matching of three quantities: optical flow, texture and also image blocks (the last one in order to deal with large translation of the head). This approach aims to exploit the benefits of the two previous methods, defining the cost functional (1.1.1) as a linear combination:

$$J = a \cdot J_{texture} + b \cdot J_{optical\ flow}$$

This choice appears to be slightly risky, since the linear combination is obtained between quantities that are not homogeneous and from the paper is not clear how the coefficients a and b are tuned and if their value is independent from the particular type of video stream. The internal model for the head is a simple ellipsoid or an head-like close surface obtained from a set of points that represent different profiles of the head that will be tracked. The application of the last model is limited to the cases where it is possible to retrieve this set of points before starting the tracking procedure. Some quantitative experiments are carried out using a set of synthesized sequences where a texture mapped ellipsoid is rendered and moved according to trajectories where the maximum variation for the angle parameters is $8.0°$. As far as the translation parameters are concerned, also in this case the values are expressed in pixels: we believe that this choice does not provide a significant insight in the performance of the algorithm unless there is a comparison with the actual dimensions of the object that is tracked. Moreover since the shape of the object that has been rendered is a real ellipsoid, one of the major problems typical for an HTS that uses a matching approach is not considered: the disagreement between the shape of the real head and the shape of the internal 3D model. Like in **[10]**, the optimization step is carried out using the simplex gradient descent technique.

The maximum error for the angle parameters in the synthesized sequences[3] appears to be lower than $2°$. The algorithm seems to be also quite robust in handling partial occlusions of the head.

[3]Once again note that the rendered model and the internal model are the same.

1.3.4. Otsuka and Ohya's approach [13]. In their approach the authors develop a real time HTS which reconstructs the pose of the head using a set of feature points that are tracked from one frame to the other by the Kanade Lucas and Tomasi algorithm. In this case the function V may be written as a matrix where each column represents the position of a feature on the image plane:

$$V = \begin{pmatrix} x_1 & \cdots & x_N \\ y_1 & \cdots & y_N \end{pmatrix}$$

The portion of the image where the features are selected is obtained using a segmentation procedure. This procedure seems to be one of the critical points of this approach: complex scenes and varying light conditions may strongly affect the performance of the segmentation procedure. The head motion is estimated from the coordinates of the features points in successive frames, by using the epipolar geometry of a weak perspective perspective model. In this papers the authors make the strong assumption that the person whose head is to be tracked is sitting on a chair in front of the camera, so that only the rotation of the head is considered: in this case the parameter vector ϕ is composed by four components: three angles and a scaling factor. The internal model for the head is a cylinder. No quantitative data to measure the accuracy of the algorithm are available.

CHAPTER 2

The Feature Point Based Head Tracking Algorithm

2.1. Introduction

The basic idea for our algorithm is similar to the one presented in Otsuka and Ohya's paper [**13**], even though there some relevant differences.

Let's consider the situation depicted in figure (2.1.1): let $p(t) = \{q_j(t) \mid 1 \leq j \leq N\}$ be a set of points features on the image plane belonging to that part of the image which represents the head we are interested to track. Let's consider one feature $q_j(t)$ belonging to the set $p(t)$ and let $q_j(t+1)$ be the position of the same feature on the image plane at time $t+1$ (i.e. $q_j(t+1) \in p(t+1)$). Let Q_j be the position of that feature on the 3D head model and let $q_{j,bp}(\phi(t+1))$ its back projection on the image plane, according to the pose of the head model described by the parameter vector $\phi(t+1)$. Finally let $d_j(\phi(t+1)) = \|q_{j,bp}(\phi(t+1)) - q_j(t+1)\|$. Then the pose of the head at frame $t+1$ is the vector $\phi(t+1)$ that minimizes the quantity $\sum_{j=1}^{N} d_j^2$. We can thus say that the algorithm is basically composed by three major procedures:

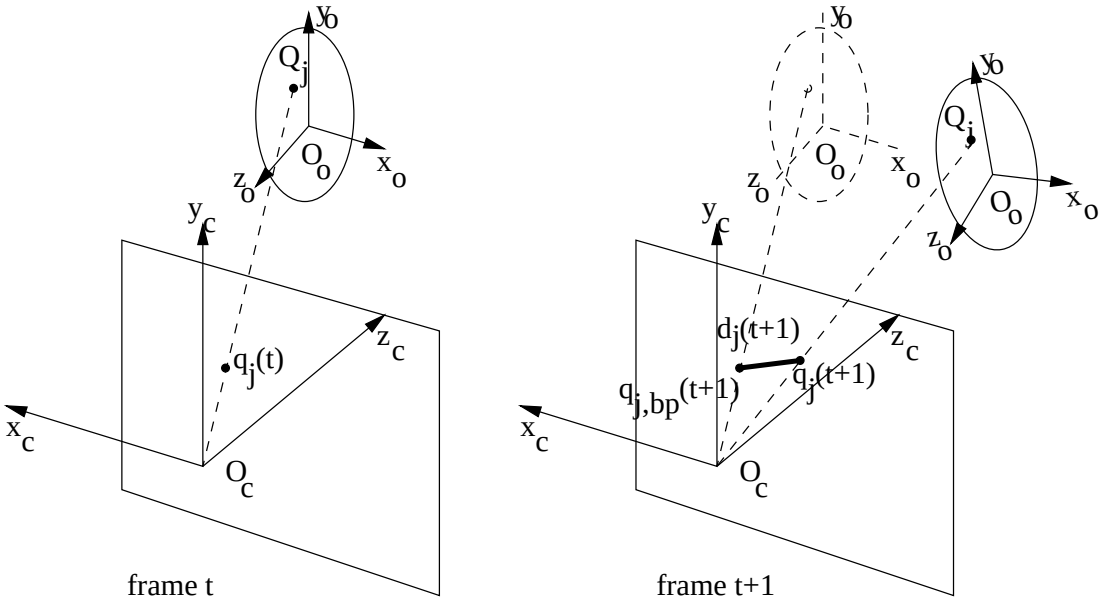

FIGURE 2.1.1. The feature tracking based method.

- a procedure that selects a set of "good" elements for the set p, and tracks them robustly from one frame to the other rejecting those elements of p no more suitable for the tracking (this procedure is represented in figure (1.1.1) by the box with the label "*computation of V from video stream*").
- a procedure that recovers an estimate of the position of the elements of the set $p(t)$ in the 3D space (let this set of points be $P = \{Q_j \mid 1 \leq j \leq N\}$) using the internal 3D model for the head and back-projects them on the image plane (this procedure is represented in figure (1.1.1) by the box with the label "*computation of V from 3D model*").
- a procedure that updates the pose of the head from frame t to frame $t+1$, trying to match the position (on the image plane) of the back projection of the set of points P with the points of the set $p(t+1)$ (this procedure is represented in figure (1.1.1) by the box with the label "*optimization*").

Now we will describe in more detail each one of the steps of the algorithm, but before we introduce the notation that will be used henceforth (see figure (2.1.2)):

- Let Σ_w, Σ_c, Σ_o be respectively the coordinate system of the world, of the camera and of the 3D head model.
- Let the subscripts *w, c, o* indicate that a point is expressed with respect to the world, camera or object coordinate system.
- Let $G_{w2c} \in SE(3)$ be[1] the transformation matrix such that: $X_c = G_{w2c} \cdot X_w$.
- Let $G_{o2w}(t) \in SE(3)$ be the transformation matrix such that: $X_w = G_{o2w} \cdot X_o$.
- Let $P = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$ the projection matrix, where f is the focal length of the camera.

2.2. 3D Model for the Head

The choice of a suitable the 3D model for the head is crucial in order to obtain a robust tracking. Basically this choice is a trade-off between two possibilities: a simple model, which is easy to fit automatically at the starting frame and that simplifies operations like image segmentation and feature projection, and a more complex model, like a 3D mesh of points closely resembling the real head structure, which on one hand improves the performance of the tracker when large

[1]The set $SE(3) = \{(t, R) \mid t \in \mathbb{R}^3, R \in SO(3)\}$ is called *special Euclidean group*, the set $SO(3) = \{R \in \mathbb{R}^{3\times 3} \mid RR^T = \mathbb{I} \wedge det(R) = 1\}$ is referred as *special orthogonal group* or *rotation group* of $\mathbb{R}^3$.

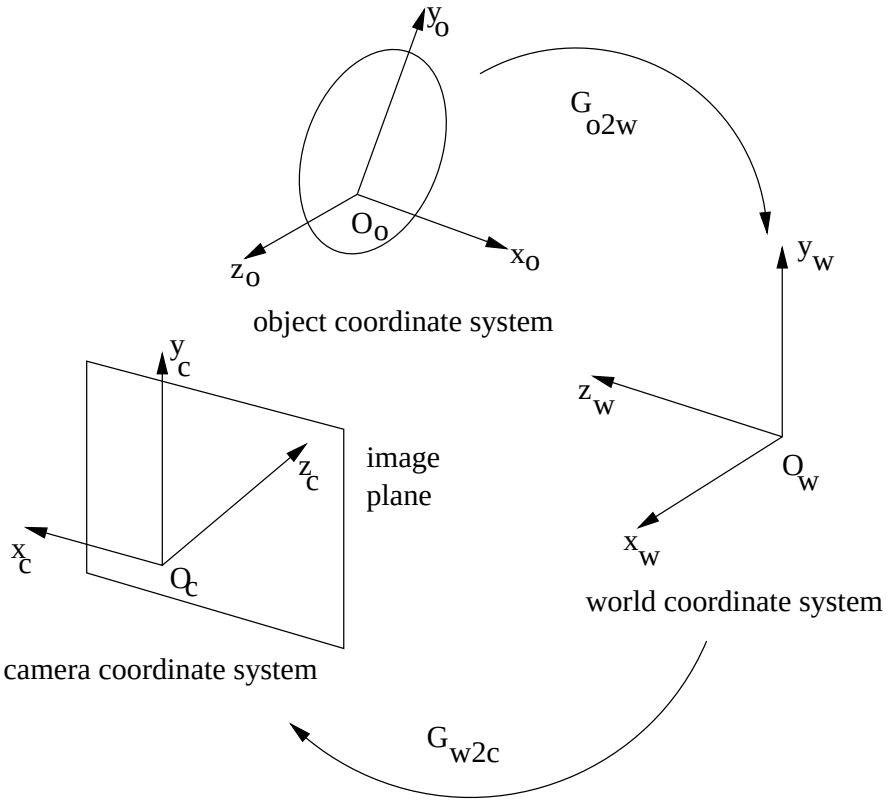

FIGURE 2.1.2. The coordinate systems with the corresponding transformation matrices.

motions are involved but on the other hand is slightly more difficult to handle and requires a more accurate fitting in the inital frame.

2.2.1. Ellipsoidal model. A 3D ellipsoid represents a fairly good approximation of the actual shape of the human head and at the same time has a compact and easy to handle analytic description. Let's consider an ellipsoid whose semiaxis are a, b and c (these are the model parameters). Using homogeneous coordinates it can be easily shown that a point $X_o = \begin{pmatrix} x_o & y_o & z_o & 1 \end{pmatrix}^T$ belongs to the surface of this ellipsoid if and only if:

$$X_o^T E_o X_o = 0 \tag{2.2.1}$$

where

$$E_o = \begin{pmatrix} b^2c^2 & 0 & 0 & 0 \\ 0 & a^2c^2 & 0 & 0 \\ 0 & 0 & a^2b^2 & 0 \\ 0 & 0 & 0 & -a^2b^2c^2 \end{pmatrix} \tag{2.2.2}$$

Moreover, since we can write $X_c(t) = G_{w2c} \cdot G_{o2w}(t) \cdot X_o$, then, with respect to the camera coordinate system, we have that:

$$X_c^T(t) E_c X_c(t) = 0$$

where

$$E_c = (G_{w2c} \cdot G_{o2w}(t))^{-T} \cdot E_c \cdot (G_{w2c} \cdot G_{o2w}(t))^{-1}$$

At this point it is necessary to explain two things: first of all why the point X_o does not depend on time, and secondly which is the reason to introduce two transformation matrix instead that just one (for example $G(t) = G_{w2c} \cdot G_{o2w}(t)$). The answer to the first question is that we assume that the deformations of the head (which may be due for example to the change of expression) are small enough so that the head can be considered a rigid body. This way the points belonging to the surface of the 3D model don't change their position with respect to the object coordinate system. As far as the second question is concerned, this choice is made to study the performance of the algorithm using more than one camera: this way the set of parameters that describe the pose of the object (with respect to the world coordinate system) are always the same.

2.2.2. Superellipsoidal model. The introduction of this model is justified by the fact that the structure of the head is some way more "squared" than the surface described just using an ellipsoid. In order to study a superellipsoidal surfaces in some more details let's first give some definitions.

DEFINITION 2.2.1. Let $g(u) : I_g \subseteq \mathbb{R} \to \mathbb{R}^2$ and $h(v) : I_h \subseteq \mathbb{R} \to \mathbb{R}^2$ be two parameterized curves: we define the *spherical product* between the two curves as it follows:

$$g(u) \otimes h(v) = \begin{pmatrix} g_x(u) \cdot h_x(v) \\ g_y(u) \cdot h_x(v) \\ h_y(v) \end{pmatrix}$$

The spherical product is an efficient way two represent a 3D surface in terms of two 2D curves. For example (see figure (2.2.1)), if $g(u)$ and $h(v)$ are the parameterized representation of an ellipse, say $g(u) = \begin{pmatrix} a_g \cos u & b_g \sin u \end{pmatrix}^T$, where $0 \leq u < 2\pi$, and $h(v) = \begin{pmatrix} a_h \cos v & b_h \sin v \end{pmatrix}^T$, where $0 \leq v < \pi$, then the spherical product between these two curves is:

$$g(u) \otimes h(v) = \begin{pmatrix} a_g a_h \cos u \cos v \\ b_g a_h \sin u \cos v \\ b_h \sin v \end{pmatrix}$$

which is the usual polar representation for a 3D ellipsoid. If we are given the

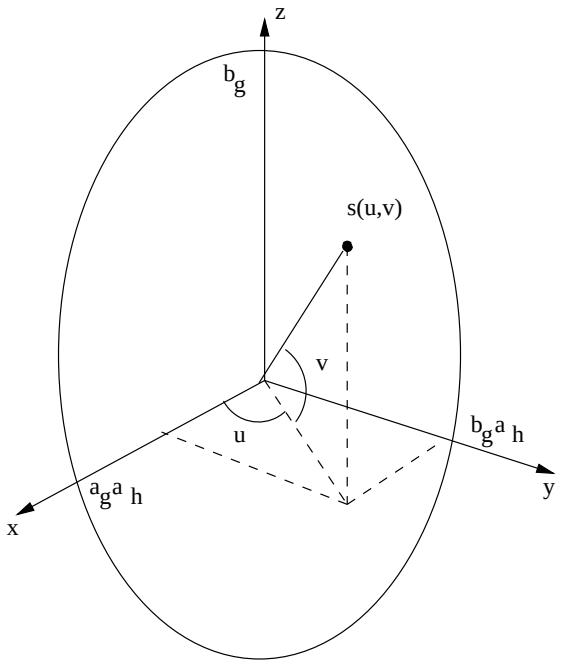

FIGURE 2.2.1. A 3D ellipsoid: the spherical product of two 2D ellipses.

analytical expression for the curves $g(u)$ and $h(v)$, it is also straightforward to recover the normal vector to the surface $s = g \otimes h$ at the point $\begin{pmatrix} u & v \end{pmatrix}^T$. First of all we observe that the tangent plane at that point is spanned by the vectors $t_1(u,v) = \frac{\partial s}{\partial u}(u,v)$ and $t_2(u,v) = \frac{\partial s}{\partial v}(u,v)$. Thus the normal can be calculated evaluating the cross product between t_1 and t_2 and normalizing the result:

$$n(u,v) = \frac{t_1(u,v) \wedge t_2(u,v)}{\|t_1(u,v) \wedge t_2(u,v)\|}$$

In this example we used the spherical product of two conics to represent a quadric. Now we will build a superquadric using the product of two superconics, in particular a superellipsoid obtained by the spherical product of two superellipses.

DEFINITION 2.2.2. The curve that is implicitly defined by the equation:

$$\left(\frac{x^2}{a^2}\right)^{\frac{1}{\gamma}} + \left(\frac{y^2}{b^2}\right)^{\frac{1}{\gamma}} = 1 \tag{2.2.3}$$

is called *superellipsoid.*

The role of the parameter y is to "inflate" or "deflate" an ellipsoid (see figure (2.2.2)). It is easy to verify that the polar parameterization for a superellipse is

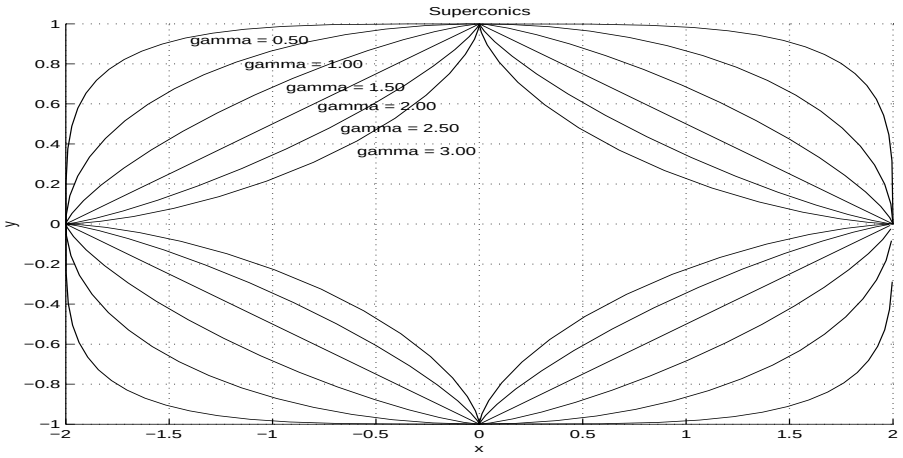

FIGURE 2.2.2. Different superellipses for different values of γ, with $a = 2$ and $b = 1$.

given by[2]:

$$g(u) = \begin{pmatrix} a \cdot sign\,(\cos u) \cdot |\cos u|^{\gamma} \\ b \cdot sign\,(\sin u) \cdot |\sin u|^{\gamma} \end{pmatrix} \tag{2.2.4}$$

Therefore to obtain a superellipsoid surface we just have to calculate the spherical product between two superellipses: one example is in figure (2.2.3). It is important to notice that if we generate the surface $s(u, v)$ using the parameterization (2.2.4), the distribution of the points on the surface is not homogeneous. In our application this is not a problem, since the density of the points is higher in those parts where the curvature of the surface is bigger, and this is enough to obtain reliable results for the ray tracing procedure. Since the equation that implicitly defines the superellipsoid is quite difficult to handle[3], for the internal representation of the model we use the set:

$$M = \left\{ \begin{pmatrix} x_i & y_i & z_i \end{pmatrix}^T \mid 1 \leq i \leq m \right\} \tag{2.2.5}$$

that contains m points that lie on the surface of the model.

2.2.3. Detailed head model. A detailed model for the human head is represented (similarly to the case of a superellipsoid described in section (2.2.2)) by a set of points that lie on the the surface of the head:

$$M = \left\{ \begin{pmatrix} x_i & y_i & z_i \end{pmatrix}^T \mid 1 \leq i \leq m \right\}$$

[2]The presence of the *sign* functions together with the module is necessary to handle the real power of a negative number. Note that when $\gamma = 1$, (2.2.4) becomes the common polar expression for an ellipse whose semiaxis are a and b.

[3]Especially when we want to transfer the feature points from the image plane to the head model using ray tracing techniques.

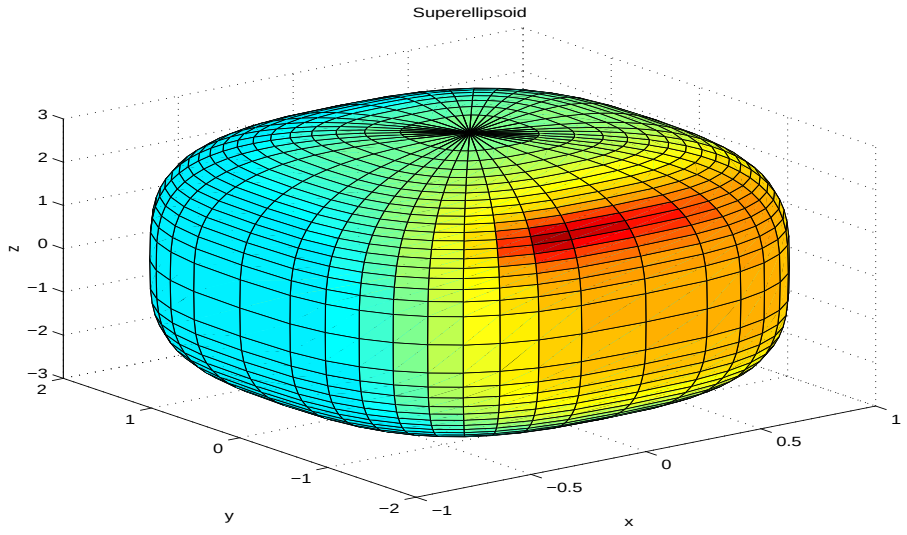

FIGURE 2.2.3. An example of a super ellipsoid with $\gamma_g = 0.6$ and $\gamma_h = 0.7$.

An example of such a model is given in figure (2.2.4).

2.2.4. Scaling the models. In order to fit the model to the actual shape of the head that will be tracked we need to modify its dimensions. As far as the ellipsoidal model is concerned, this is an easy task, since we just have to modify the values of the semiaxis a, b and c in the matrix E_o (see (2.2.2)).

When we have models that are represented through the set (2.2.5) things become slightly more complicated. Scaling the mesh of points along the three axis of the coordinate system Σ_o is simple: we just have to multiply each of the points in the set M by a diagonal matrix:

$$S = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix}$$

where s_x, s_y and s_z are the scaling factors along the three axis. But what happens to the normals to the surface at those points? The problem is that the scaling operation does not preserve the angles. Let's consider the situation depicted in figure (2.2.5): a surface patch may be approximated by the triangle whose vertex are the points P, N_1 and N_2 and consequently the normal to the surface at the point P may be approximated by the unit vector:

$$n(P) = \frac{N_1P \wedge N_2P}{\|N_1P \wedge N_2P\|}$$

What happens to the normals when we scale the model according the matrix S? The answer is given by the following lemma:

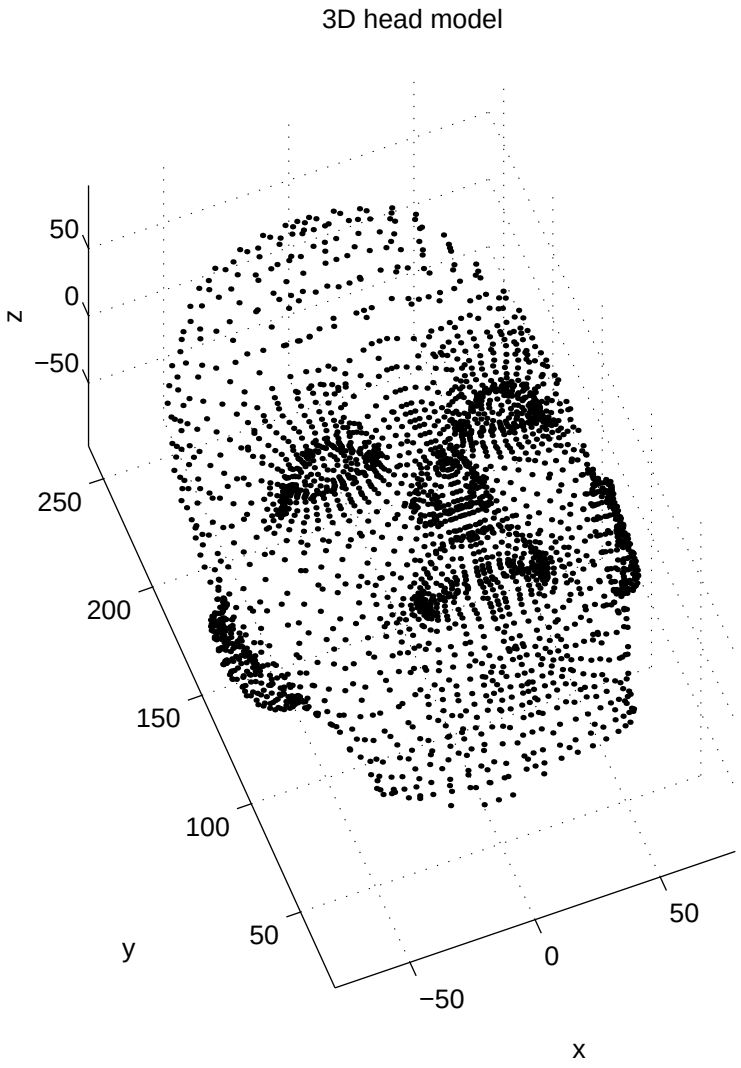

FIGURE 2.2.4. The set of points that represent the 3D head model.

LEMMA 2.2.3. *Let u_1 and u_2 be two vectors in $\mathbb{R}^3$ and let $S = diag\,(\alpha, \beta, \gamma)$. Then:*

$$Su_1 \wedge Su_2 = \begin{pmatrix} \beta y & 0 & 0 \\ 0 & \alpha y & 0 \\ 0 & 0 & \alpha\beta \end{pmatrix} [u_1 \wedge u_2] \tag{2.2.6}$$

PROOF. Let's write explicitly the vector product between Su_1 and Su_2:

$$Su_1 \wedge Su_2 = \begin{vmatrix} \alpha u_{1,x} & \beta u_{1,y} & \gamma u_{1,z} \\ \alpha u_{2,x} & \beta u_{2,y} & \gamma u_{2,z} \\ i & j & k \end{vmatrix} = \begin{pmatrix} \beta\gamma \cdot (u_{1,y}u_{2,z} - u_{1,z}u_{2,y}) \\ \alpha\gamma \cdot (u_{1,z}u_{2,x} - u_{1,x}u_{2,z}) \\ \alpha\beta \cdot (u_{1,x}u_{2,y} - u_{1,y}u_{2,x}) \end{pmatrix} \tag{2.2.7}$$

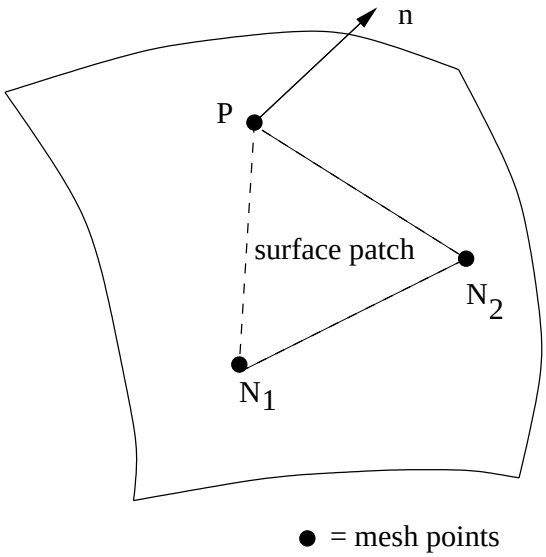

FIGURE 2.2.5. The normal to a point for surfaces represented using a set of points.

Since

$$u_1 \wedge u_2 = \begin{pmatrix} u_{1,y}u_{2,z} - u_{1,z}u_{2,y} \\ u_{1,z}u_{2,x} - u_{1,x}u_{2,z} \\ u_{1,x}u_{2,y} - u_{1,y}u_{2,x} \end{pmatrix} \tag{2.2.8}$$

comparing (2.2.7) with (2.2.8) we can deduce the equivalence stated in the lemma. □

Thus after the scaling of the point mesh we just have to apply (2.2.6) and normalize the result.

2.3. Point Selection, Tracking and Rejection

2.3.1. Point selection. The elements of the set $p(t)$ (defined in (2.1)) must satisfy two basic requirements: first of all they must belong to that portion of the image that represents the head, and secondly they must be robustly tracked from one frame to the other.

To solve the first problem we must be able to segment the area of the image that represents the head. A general solution for this problem is quite difficult to implement without doing any sort of assumptions as far as the structure of the scene is concerned (in particular for the lighting conditions and for the background). As far as our algorithm is concerned, a reasonable (in the sense that it represents a good compromise between accuracy and computational effort) solution for this problem is obtained using the information provided by the internal 3D model of the head. Basically the idea is to project on the image plane the

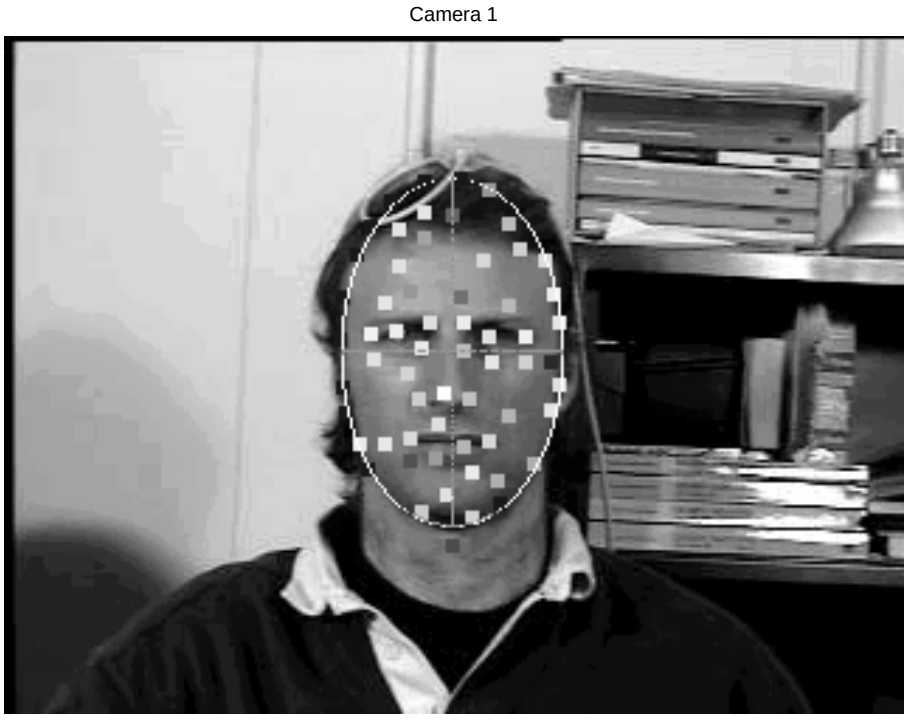

FIGURE 2.3.1. The apparent γ contour of an ellipsoid and the set of features selected by the tracker (the square windows with different gray levels).

contour Γ of the 3D model (which is a closed curve γ) and consider all the pixels inside γ as pixels belonging to the head. The problem of retrieving the curve γ is thus directly connected to the choice of the 3D model for the human head. Figure (2.3.1) shows the apparent contour γ and the features that have been selected inside this contour.

2.3.1.1. *Ellipsoidal model.* Let's give some definitions and results (see: **[1]** and **[14]**) that will allow us to recover γ when we use the ellipsoidal model described in (2.2.1):

DEFINITION 2.3.1. The *contour generator* Γ of any surface M relative to a point O is the curve of points $\Gamma \in M$ where the tangent plane passes through O. The corresponding image *apparent contour* γ is the projection of Γ on the image plane.

PROPOSITION 2.3.2. *Let E^* be the dual*[4] *of the ellipsoid E. Then, under the camera projection matrix P, the apparent contour γ is an ellipse, whose points satisfy the relation*

$$x^T C^* x = 0 \tag{2.3.1}$$

[4]If the ellipsoid is not singular (that is to say $det(E) \neq 0$), then $E^* = adj(E) = det(E) \cdot E^{-1}$.

where $C^* = PE^*P^T$.

PROOF. In order to prove this result we need to carry out two steps: first all all we need to prove that the apparent contour γ is a conic and finally that the points that lie on that conic satisfy the relation (2.3.1).

Let $X \in \mathbb{R}^3$; for each point Q that lies onto the line passing both through the optical center and through X there exists a scalar $\lambda \in \mathbb{R}$ such that the following equation holds:

$$Q = \lambda X + O \tag{2.3.2}$$

Therefore the contour generator Γ is given by all those points Q such that the discriminant of the second degree equation obtained plugging (2.3.2) in $Q^T EQ = 0$ is zero. The discriminant for such equation is given by:

$$\Delta = 4\left[\left(O^T EX\right)^2 - \left(X^T EX\right)\left(O^T EO\right)\right]$$

and, in order to have $\Delta = 0$, the following relation must be satisfied:

$$X^T\left[EOO^T E - O^T EOE\right]X = X^T SX = 0$$

This equation constrains the points X to lie on a quadric surface S. Since the interception of a quadric with a plane is always a conic we can state that the apparent contour obtained projecting Γ on the image plane is a conic. This proves the first part of the proposition.

We will now show that the points on γ satisfy the relation (2.3.1). Since a line in $\mathbb{R}^2$ is represented by an equation such:

$$ax + by + c = 0$$

we may equivalently use the vector $l = \begin{pmatrix} a & b & c \end{pmatrix}^T$ to identify that line. Moreover we have that the set of points $X \in \mathbb{R}^3$ mapping to a line l via the camera projection matrix P are all those points such that:

$$X^T P^T l = 0$$

This can be easily proved observing that a point $x = PX$ lies on the line l if and only if $x^T l = X^T P^T l = 0$. Therefore the vector $P^T l$ identifies a plane in $\mathbb{R}^3$ (this plane is often referred also as the *pull-back* of the line l). Since all the planes tangent to the ellipsoid E and passing through the point O must satisfy the relation:

$$\left(P^T l\right)^T E^*\left(P^T l\right) = l^T\left(PE^*P^T\right)l = 0$$

and since we know that the line l is tangent to a conic C, then we can also write:

$$l^T C^* l = l^T\left(PE^*P^T\right)l = 0$$

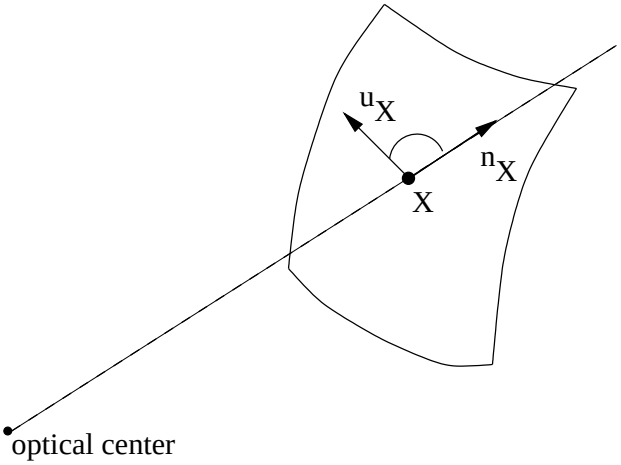

FIGURE 2.3.2. How to calculate if a point is visible from the camera.

□

Using this result we can easily compute γ, and consequently decide which is the area of the image corresponding to the head.

2.3.1.2. *Superellipsoidal model and detailed head model.* In this case the apparent contour γ cannot be recovered analytically, since either the analytical description of the surface is not simple or we do not have an analytical description at all. To solve this problem we consider the set of points m_V obtained by projecting on the image plane the elements of the set $M_V \subseteq M$, that is the subset of all the visible points belonging to M. To verify if a point $X \in M$ is visible from the camera we consider the scalar product between the unit vector that represent the direction of the ray from the optical center to M (say u_X) and the normal to the surface at point X (say n_X): if $u_X^T \cdot n_x \leq 0$ (that is to say the cosine of the angle between the two units vectors), then the point is visible (see figure (2.3.2)). Once we have determined the set of points m_V we define the apparent contour γ as the shortest curve on the image plane that contains all the points in m_V. Note that in the case of the superellipsoidal model this curve coincides with the convex hull of the set m_V.

2.3.2. Feature tracking. Now we have to choose among the points inside the contour γ those who can be robustly tracked and track them from one frame to the other. To achieve this goal we used the feature tracking method suggested by Kanade, Lucas and Tomasi **[2][3]**. The idea is based on the *brightness change constraint equation* (BCCE): let's consider the pixel at the position $\begin{pmatrix} x & y \end{pmatrix}^T$ and let $I(x, y)$ be the luminance of that pixel; let's also suppose that the pixel moves from the position $\begin{pmatrix} x & y \end{pmatrix}^T$ at frame t to the position $\begin{pmatrix} x+u & y+v \end{pmatrix}^T$

at frame $t+1$. If the frame rate is high enough, then the luminance of two corresponding pixels should not change between one frame and the other. This constraint can be expressed by the following equation:

$$I(x+u, y+v, t+1) = I(x, y, t) \tag{2.3.3}$$

If we expand the first member of the previous equation using Taylor expansions we obtain:

$$I(x+u, y+v, t+1) \cong I(x, y, t) + I_x(x, y, t)u_x + I_y(x, y, t)u_y + I_t(x, y, t) \tag{2.3.4}$$

where I_x, I_y and I_t are respectively the gradients and the temporal derivative of the image. Equating (2.3.3) and (2.3.4) we obtain:

$$\begin{pmatrix} I_x(x, y, t) & I_y(x, y, t) \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} + I_t(x, y, t) = 0 \tag{2.3.5}$$

We want to use this equation to estimate the value of $\begin{pmatrix} u & v \end{pmatrix}^T$. Instead of just considering one single pixel we may consider a window W_k and estimate which is the translation of that window. Thus the (2.3.5) can be written as:

$$\begin{pmatrix} \sum_{W_k} I_x(x, y, t) & \sum_{W_k} I_y(x, y, t) \end{pmatrix} \begin{pmatrix} u_k \\ v_k \end{pmatrix} + \sum_{W_k} I_t(x, y, t) = 0 \tag{2.3.6}$$

The problem can now be restated as an optimization problem: if we introduce the cost functional:

(2.3.7)

$$J(u, v) = \left[\begin{pmatrix} \sum_{W_k} I_x(x, y, t) & \sum_{W_k} I_y(x, y, t) \end{pmatrix} \begin{pmatrix} u_k \\ v_k \end{pmatrix} + \sum_{W_k} I_t(x, y, t) \right]^2$$

then the optimal displacement vector that describes the new position of the window W_k is given by:

$$\begin{pmatrix} u_k^* \\ v_k^* \end{pmatrix} = arg \min_{\begin{pmatrix} u \\ v \end{pmatrix} \in \mathbb{R}^2} J(u, v)$$

To solve this problem we take the derivatives of J with respect to u and v, and we equate them both to zero: we obtain a linear equation in the form of:

$$C_k \cdot \begin{pmatrix} u_k \\ v_k \end{pmatrix} + D_k = 0 \tag{2.3.8}$$

where:

$$C_k = \begin{pmatrix} \sum_{W_k} I_x^2 & \sum_{W_k} I_x I_y \\ \sum_{W_k} I_x I_y & \sum_{W_k} I_y^2 \end{pmatrix} \qquad D_k = \begin{pmatrix} \sum_{W_k} I_x I_t \\ \sum_{W_k} I_y I_t \end{pmatrix}$$

The solution for the displacement is (if C_k is not singular):

$$\begin{pmatrix} u_k^* \\ v_k^* \end{pmatrix} = -C_k^{-1} \cdot D_k \tag{2.3.9}$$

Observing the previous expression we can now also answer the question: "which points should be chosen?"[5] The key observation is that the matrix C_k should not be ill-conditioned, so that it could be inverted robustly. To ensure that we check the spectrum $\sigma(C_k)$: a well conditioned (square) matrix is a matrix whose eigenvalues are not close to zero and that have similar orders of magnitude. Since the maximum value of a each component of C_k is bounded (the luminance function can take values only on a finite interval...), then also the maximum value for the eigenvalues of that matrix is bounded. Then we will say a window W_k to be good whenever $\min \sigma(C_k) > \lambda_T$, where λ_T is an appropriate threshold value. Note that the selection criterion is optimal by construction: a good feature is one that can be tracked well.

To improve the performance of the tracking algorithm the expression (2.3.9) can be used in an iterative fashion which resembles the Newton Raphson algorithm: once we have a first estimate of the displacement vector we rewarp[6] the image of the frame t according to this estimate and repeat the minimization process (updating the expressions for C_k and D_k). This allows the algorithm to obtain a subpixel accuracy in the estimation of the displacement vector.

Now we would like to spend a few words about the choice of the dimensions of the window W_k; the two key components for the feature tracker are accuracy and robustness. A small window would be preferable in order not to "smooth out" the details contained in the image. On the other hand, to handle large motions, possibly also under varying lighting conditions, it is preferable to pick a large integration window.

In order to deal with large displacements it can be useful to use a pyramidal version of the Kanade Lucas and Tomasi algorithm [**4**]: the central motivation behind pyramidal representation is to be able to handle *large* pixel motions (where large means that the norm of the displacement vector is larger then the size of the window).

[5]Since we are considering a window rather than just a point, the coordinates of the feature that will be tracked correspond to the center of the window.

[6]Since the values for u_k and v_k in general are not integer we need to carry out a sub-pixel image interpolation.

DEFINITION 2.3.3. Let $I : [1,m] \times [1,n] \subseteq \mathbb{N}^2 \to [0,1]$ be the image matrix: then jth level of the pyramid representation of the image is defined by the following recursion[7]:

$$I^j(x,y) = \begin{cases} I(x,y) & j = 0 \\ \frac{1}{4} I^{j-1}(2x,2y) + & \\ \frac{1}{8}\left[I^{j-1}(2x-1,2y) + I^{j-1}(2x+1,2y) + \right. & \\ \left. I^{j-1}(2x,2y-1) + I^{j-1}(2x,2y+1)\right] + & j \neq 0 \\ \frac{1}{16}\left[I^{j-1}(2x-1,2y-1) + I^{j-1}(2x+1,2y+1) + \right. & \\ \left. I^{j-1}(2x+1,2y-1) + I^{j-1}(2x-1,2y+1)\right] & \end{cases}$$

The algorithm now proceeds as follows: first the displacement vector (2.3.9) is computed at the deepest pyramid level j_{max} (in the Newton-Raphson fashion described before). Then the result is propagated to the upper level $j_{max} - 1$ through the simple relation:

$$\begin{pmatrix} u_k^{j-1} \\ v_k^{j-1} \end{pmatrix} = 2 \cdot \begin{pmatrix} u_k^j \\ v_k^j \end{pmatrix}$$

and it is used as an initial guess for the estimation of the displacement vector at this level. The algorithm is iterated until we reach the zero level of the pyramid. Which is the advantage of this implementation? It becomes clear when we observe that the final displacement vector (at the zero level of the pyramid) is given by:

$$\begin{pmatrix} u_k \\ v_k \end{pmatrix} = \sum_{j=0}^{j_{max}} 2^j \cdot \begin{pmatrix} u_k^j \\ v_k^j \end{pmatrix} \tag{2.3.10}$$

If the basic implementation of the algorithm can handle a displacement vector where the maximum absolute value for each component is Δ, then the pyramidal implementation allows us to handle larger displacements; more precisely, using (2.3.10), we obtain:

$$\Delta_{max} = \sum_{j=0}^{j_{max}} 2^j \cdot \Delta = (2^{j_{max}+1} - 1) \cdot \Delta$$

2.3.3. Feature rejection. The causes that may lead a feature to be discarded by the algorithm are basically two:

(1) Bad tracking quality

(2) Excessive back projection disagreement

Before giving a closer look to each one of these two possibilities it is important to notice that a feature remains "marked" on the internal head model for all

[7]Special care must be used to define recursion at the boundaries of the image: when the index is out of bounds we choose the closest "in bounds" index.

the frames before its rejection. This fact makes the algorithm more robust with respect to the cumulative tracking error that may be introduced because of the model disagreement or because of the presence of outliers.

2.3.3.1. *Bad tracking quality.* We say that the tracking quality of the jth feature is bad when the value of the cost functional (2.3.7) for the optimal value of the displacement vector (2.3.9) is greater or equal than a fixed threshold J_T:

$$J\left(u_j^*, v_j^*\right) \geq J_T$$

There are many different reason that may cause the residual of the cost functional to be higher than the threshold J_T. First of all varying light conditions may change the pixel pattern of the window W_j, so that two windows at two consecutive frames corresponding to the same part of the head may heavily differ. Secondly the portion of the image contained in the window W_j may undergo some sort of distortion due to the perspective projection of a 3D object onto a 2D plane: also in this case the content of the window in two consecutive frames may be quite different.

2.3.3.2. *Excessive back projection disagreement.* This happens when the distance between the back-projected feature $q_{j,bp}\left(\phi(t+1)\right)$ and the feature tracked at frame $t+1$ (i.e. $q_j(t+1)$) (*after* the optimization process is ended) is greater than a certain threshold d_T:

$$d_j\left(\phi(t+1)\right) = \left\| q_{j,bp}\left(\phi(t+1)\right) - q_j\left(t+1\right) \right\| \geq d_T$$

This happens to be true for features that are marked in places where the disagreement between the actual shape of the head and the shape of the 3D model is more relevant. In this case the point Q_j is back projected in a point that does not coincide with the position of the tracked feature: this situation is depicted in figure (2.3.3) where for sake of convenience one dimension has been removed.

Another fact that may cause d_j to be greater than the threshold value is when the corresponding feature does not belong to the portion of the image that represents the head, even though it is internal to the apparent contour γ. In this case the trajectory of the feature tracked on the image plane is no more coherent with the trajectory of the feature back projected from the 3D model: this will cause d_j to exceed the threshold value and consequently the feature will be rejected. This rejection criteria makes the tracking algorithm quite robust in the presence of outliers.

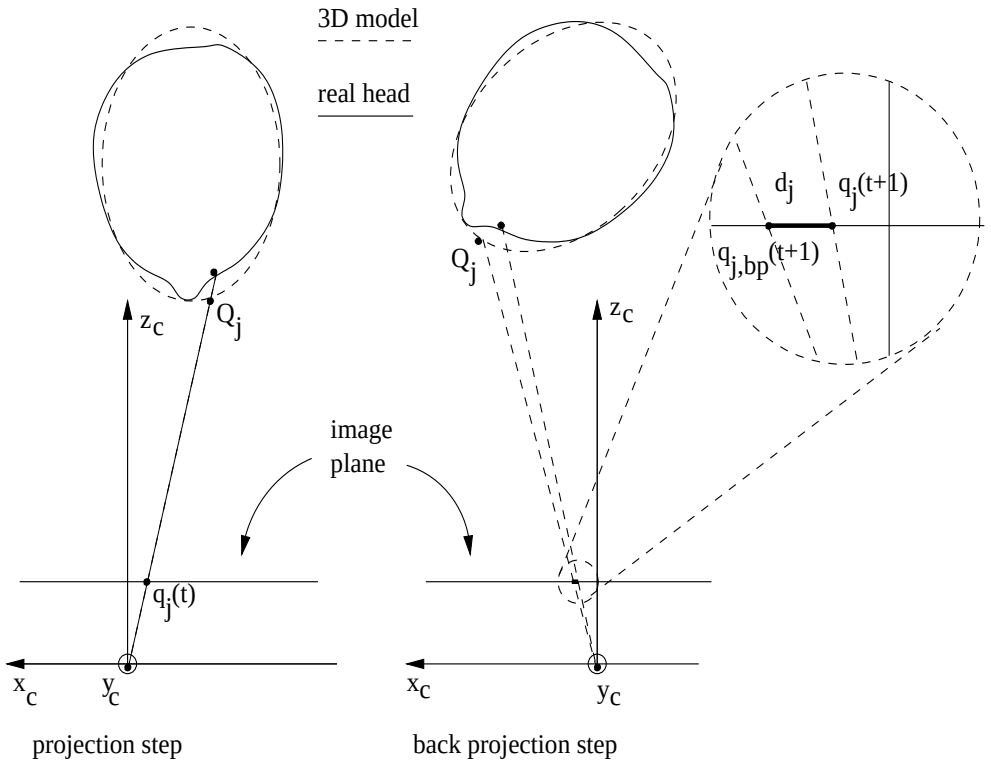

FIGURE 2.3.3. The model disagreement error.

2.4. Construction of the set P

To construct this set we assume that we have the set $p(t)$ and an estimate of the matrix $G_{o2w}(t)$. The procedure to construct the set P is based on a ray-tracing technique: let's consider a point $q_j(t) \in p(t)$ and let's consider the line l in the 3D space from the optical center O_c with direction $t_c = \begin{pmatrix} q_j(t) \\ f \end{pmatrix} \in \mathbb{R}^3$. The point Q_j is constrained to lie on this line: thus its position can be recovered intersecting this line with the surface of the model that represents the head. The exact calculation of Q_j is easy for surfaces that have a simple analytical description, as in the case of an ellipsoid, whereas things become more complicated both for a superellipsoid (whose analytical description does not allow to express explicitly the intersection of such a surface with a line) and for a real head model (where an analytical representation is not available at all).

2.4.1. Ellipsoidal model. In the object coordinate system a generic point that lie on the ray from the optical center through the feature point q_j on the image plane can be expressed by:

$$X_o(\lambda) = \lambda t_o + O_o \tag{2.4.1}$$

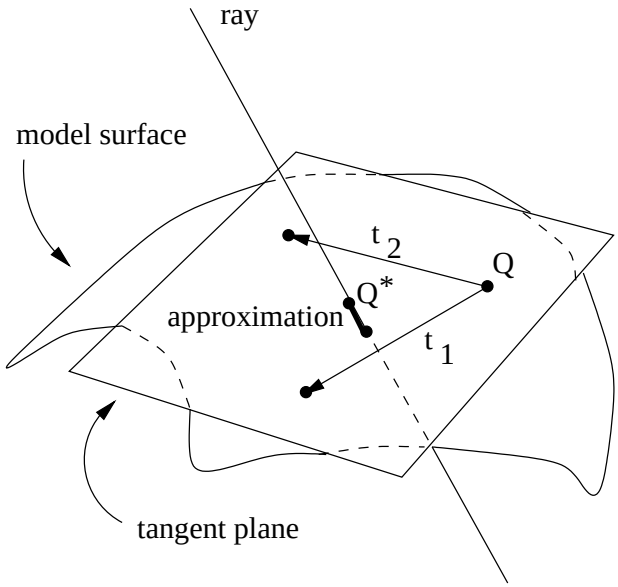

FIGURE 2.4.1. Ray tracing procedure for the superellipsoidal model and the detailed head model.

In order to recover the intersection of the line described by (2.4.1) with the surface of the ellipsoid, we plug (2.4.1) in (2.2.1), obtaining a simple second degree equation $a\lambda^2 + b\lambda + c = 0$, where:

$$a = t_o^T \cdot E_o \cdot t_o$$

$$b = 2 \cdot t_o^T \cdot E_o \cdot O_o$$

$$c = O_o^T \cdot E_o \cdot O_o$$

The roots of this equation may be of three different types according to the value of $\Delta = b^2 - 4ac$:

- $\Delta > 0$) we have two different real roots since the line l hits the ellipsoid in two different points (the entry point and the exit point). We just have to choose the minimum value of λ, that is associated to the entry point.
- $\Delta = 0$) we have two real and coincident roots: the line l is tangent to the ellipsoid surface.
- $\Delta < 0$) we have two complex conjugate roots: the line does not hit the ellipsoid.

2.4.2. Superellipsoidal model and detailed head model. In this case we do not have analytical description of the surface, but only a set of points $M = \left\{ \begin{pmatrix} x_i & y_i & z_i \end{pmatrix}^T \mid 1 \leq i \leq m \right\}$ that lie on that surface. The basic idea is to approximate the surface near the interception point with its tangent plane. Let's consider the situation depicted in figure (2.4.1). The first step we carry

out is to determine the closest point to the ray (2.4.1) belonging to the set of the visible points $M_V \subseteq M$. To do that we first compute the distance of all the points in M_V from the ray (2.4.1) using the following lemma:

LEMMA 2.4.1. *Let $Q_o \in M_V$ and let's consider the line parameterized in (2.4.1). Then the minimum distance between the point Q_o and that line is given by:*

$$d = \left\| u_o - t_o^T u_o t_o \right\|$$

where $u_o = Q_o - O_o$ and t_o is the unit vector that represent the direction of the ray from the optical center of the camera through the correspondent feature on the image plane with respect to the object coordinate system.

PROOF. The squared distance between a point $X_o(\lambda)$ on the ray (2.4.1) and the point Q_o is given by:

$$d(\lambda)^2 = \|\lambda t_o + O_o - Q_o\|^2 = \|\lambda t_o - u_o\|^2 = (\lambda t_o - u_o)^T (\lambda t_o - u_o)$$

In order to find the value $\lambda^* = arg\ \min_{\lambda \in \mathbb{R}} d(\lambda)^2$ we just take the derivative of the previous expression and equate it to zero:

$$\frac{d}{d\lambda} d(\lambda)^2 = 0$$

Thus we obtain $\lambda^* = \frac{t_o^T u_o}{t_o^T t_o} = t_o^T u_o$ since $\|t_o\| = 1$. The minimum distance between the ray and the point Q_o is then:

$$d = d(\lambda^*) = \left\| t_o^T u_o t_o - u_o \right\|$$

□

Now let's consider the closest points Q to the ray (2.4.1). The set of the points that form the tangent plane to the model surface at point Q is:

$$T = \left\{ X \in \mathbb{R}^3 \mid n^T X = 0 \right\}$$

where n is the normal of the model surface in Q. The interception between the ray (2.4.1) and T is the point $X(\lambda^*)$, where:

$$\lambda^* = -\frac{n_o^T O_o}{n_o^T t_o}$$

is the solution of the equation $n^T X(\lambda) = 0$.

2.5. Recovering the Pose

At this point we are given the following data:

- the set of points $p(t+1)$.
- the set of points P.

- the pose of the human head at time t (i.e. $G_{o2w}(t)$), obtained at the previous iteration.

We now want to use these set of data to estimate $G_{o2w}(t+1)$. As said in the introduction (2.1), the basic idea is to find the $SE(3)$ matrix that minimizes the distance between the back projection of the points in P and the corresponding points of the set $p(t+1)$. First of all we must choose a convenient parameterization for the matrix $G_{o2w} = \begin{pmatrix} R & T \\ 0 & 1 \end{pmatrix} \in SE(3)$. In fact for the optimization algorithm not all the parameterizations of R are equivalent . The best performance has been obtained using a local parameterization of $SO(3)$ given by the ZYX Euler angles; the 6 parameters that describe G_{o2w} can be collected in the vector $\phi = \begin{pmatrix} t_x & t_y & t_z & \omega_x & \omega_y & \omega_z \end{pmatrix}^T$. Thus we have that:

$$R = R_z(\gamma) \cdot R_y(\beta) \cdot R_x(a) \qquad T = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

Let $\widehat{G}_{o2w}(t+1)$ be an estimate of $G_{o2w}(t+1)$ and let $Q_j \in P$; then the homogeneous coordinates of the back-projection of Q_j on the image plane at time $t+1$ are given by the expression:

$$q_{j,bp}\left(\widehat{\phi}(t+1)\right) = \mathcal{C}\left[P \cdot G_{w2c} \cdot \widehat{G}_{o2w}(t+1) \cdot Q_j\right]$$

where $\mathcal{C} : \begin{pmatrix} x & y & z \end{pmatrix}^T \mapsto \begin{pmatrix} \frac{x}{w} & \frac{y}{w} \end{pmatrix}^T$ is the *nonlinear*[8] map that converts homogeneous coordinates into Cartesian coordinates. We can thus introduce the cost functional:

(2.5.1)

$$J\left(\widehat{\phi}(t+1)\right) = \sum_{j=1}^{N} w_j d_j^2\left(\widehat{\phi}(t+1)\right) = \sum_{j=1}^{N} w_j \left\| q_{j,bp}\left(\widehat{\phi}(t+1)\right) - q_j(t+1)\right\|^2$$

which measures the disagreement between the back-projection of the points of the set P using the matrix $\widehat{G}_{o2w}(t+1)$ and the points tracked from frame t to frame $t+1$. The scalar w_j weights the contribution of the feature j in the global cost. The value for the pose parameter vector at time $t+1$ is thus given by:

$$\phi(t+1) = arg \min_{\phi \in \mathbb{R}^6} J(\phi)$$

The starting point for the optimization algorithm is chosen to be the estimation of the pose parameter vector at the previous frame.

[8]Note that what causes this problem to be a nonlinear least squares problem is the fully perspective model used for the camera geometry.

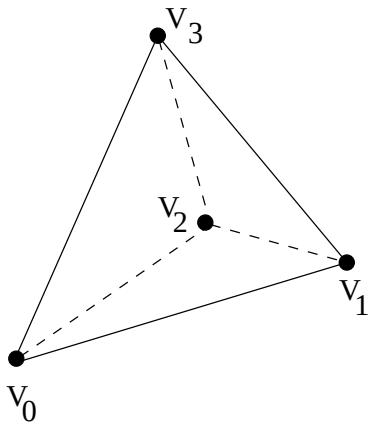

FIGURE 2.5.1. A simplex in a three dimensional space.

To find the minimum of the cost functional J we tried different algorithms: a genetic algorithm, the Levenberg and Marquardt's algorithm, and the downhill simplex method proposed by Nelder and Mead. The genetic algorithm proved to be quite accurate, robust in presence of noise and outliers features and able to find the absolute minimum of J. Unfortunately this algorithm is very expensive under a computational point of view and a real-time implementation seems not to be feasible. The Levenberg Marquardt's algorithm is faster than the genetic algorithm, but there are two major drawbacks: first of all it's necessary to compute the Jacobian of the cost functional J; the explicit calculation may provide very complex and messy formulas that could negatively affect the performance of the algorithm because of the finite precision and the round off errors introduced by the computer. Secondly this algorithm does not perform very well in presence of large residuals of J. The downhill simplex method is a good compromise between accuracy and speed. On one hand only few evaluations of the cost functional are required at each optimization step and no calculation of the Jacobian is necessary. On the other hand the algorithm seem pretty robust with noisy data and it is not affected by large residuals of J. Let's now give a close look to this algorithm **[3]**[**4**].

2.5.1. The downhill simplex method. First of all let's define what is a simplex in a n-dimensional Euclidean space:

DEFINITION 2.5.1. An *n-dimensional simplex* (see figure (2.5.1)) is a geometrical figure composed by $n+1$ vertex V_j, $0 \leq j \leq n$ such that[9]:

$$\mathbb{R}^n = span\left\{V_i - V_j \mid 0 \leq i \neq j \leq n\right\}$$

[9]This implies that the simplex is non degenerate, that is to say it encloses a finite inner n-dimensional volume.

The simplex minimization algorithm attempts to minimize a cost functional $J : \mathbb{R}^n \to \mathbb{R}$, by recursively updating the position of the vertices that define an n-dimensional simplex, so that the function J evaluated at those points decrease.

The first step that must be carried out by the algorithm is the initialization of an n-dimensional simplex: given a starting point V_0 the other n points are defined through the following equation:

$$V_i = V_0 + \lambda_i e_i \qquad 1 \leq i \leq n$$

where $e_1, \ldots, e_n$ is the canonical base for $\mathbb{R}^n$ and the multipliers λ_i are determined according to the characteristic range of the corresponding variable.

At each step of the optimization process a new point inside or near the current simplex is generated. The value of J at this new point is compared with the values of J at the vertices of the current simplex and usually one of the vertices is substituted by that new point. The algorithm iterates until the diameter of the simplex is less than a specified threshold value. In order to avoid this termination criteria to be fooled by a single anomalous step, the algorithm is restarted with a simplex constructed around the point that has been found to be the optimal solution. This further optimization step it is not expensive: if the initial point is actually the optimal point, the algorithm will stop just after a few steps. Let's now give a few definitions in order to explore in more detail the operations used to create a new point for the simplex: let $V_l = arg\ \min_{V_i} J(V_i)$, $V_h = arg\ \max_{V_i} J(V_i)$ and $\overline{V} = \frac{1}{n}\sum_{j=1}^{n} V_j$. The basic four operations through which the simplex is updated are:

(1) *Reflection*: with this operation we move away from the vertex V_h:

$$V_{new} = \overline{V} + \alpha(\overline{V} - V_h) \qquad \alpha > 0$$

(2) *Expansion*: with this operation we expand the simplex at the vertex V_i:

$$V_{new} = \overline{V} + \beta(V_i - \overline{V}) \qquad \beta > 0$$

(3) *Contraction*: the simplex is contracted at the vertex V_h:

$$V_{new} = \overline{V} + \gamma(V_h - \overline{V}) \qquad 0 < \gamma < 1$$

(4) *Shrinkage*: each of the vertices of the simplex shrinks:

$$V_i = \frac{1}{2}(V_i + V_l)$$

Finally a few words about the convergence properties of the algorithm: even though as far as now there isn't a proof that shows that the algorithm converges

to a minimizer, the results obtained in practice seem to be very good. The situation is well described by the following paragraph from **[15]**:

> Our general conclusion about the Nelder-Mead algorithm is that the main mystery to be solved is not whether is ultimately convergent to a minimizer-for general (non convex) functions, it does not-but rather why it tends to work so well in practice by producing a rapid initial decrease in function values.

2.5.2. Filtering the results. The retrieval of the quantity V (defined in section (1.1)) directly from the video stream is usually affected by noise. As described in section (2.3.3.1), two of the principal causes that may negatively affect the performance of the Kanade, Lucas and Tomasi tracking algorithm are the varying light conditions, and the perspective distortions. The effect of a bad estimation of the trajectory of the features points on the image plane affects the optimization procedure: the estimated pose parameter vector $\widehat{\phi}$ is in general affected by noise. In order to filter the data produced by the optimization procedure we decided to implement a discrete Kalman filter for each component of the pose parameter vector ϕ.

The internal model of the system is a constant acceleration model: for each of the components of ϕ we have that the state update equation is:

$$\begin{pmatrix} \phi_i(t+1) \\ v_i(t+1) \\ a_i(t+1) \end{pmatrix} = \begin{pmatrix} 1 & \Delta T & 0 \\ 0 & 1 & \Delta T \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \phi_i(t) \\ v_i(t) \\ a_i(t) \end{pmatrix} + w(t)$$

where ΔT is the frame rate, and the measurement equation is:

$$\widehat{\phi_i}(t+1) = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \phi_i(t) \\ v_i(t) \\ a_i(t) \end{pmatrix} + v(t)$$

where $w \in \mathcal{WGN}(0, Q)$, $v \in \mathcal{WGN}(0, R)$.

2.6. Algorithm parameters

In this section we will examine some of the issues related to implementation of the algorithm, in particular the tuning of those quantities that are related to the architecture of the algorithm itself.

2.6.1. Number of the features tracked. The first parameter that we take into consideration is the number of the features N that we wish to track. If, on one hand, a greater number of features may cause the optimization process to be more precise and insensitive to noise and outliers, on the other hand the time

required to process each frame is improved as well. Thus there is a trade-off between accuracy and speed of the algorithm.

We should also notice that it is not guaranteed that the tracking algorithm is able to track the specified number of the features along the whole video sequence. This is mainly due to the fact that, as described in (2.3.2), the minimum eigenvalue for each of the windows W_k should be greater than a threshold value λ_T. If the area of the image corresponding to the head is for some reason poor in texture it may be not possible to find N features inside the contour γ that satisfy the constraint of the spectrum of the matrix C_k. Moreover, since the distance between the features is required to be greater than a minimum distance d_{min}, this constraint may further reduce the number of features that can be selected and tracked.

A set of tests to compare the influence of different values for this parameter on the global performance of the tracking procedure has been carried out in (3.1.3).

2.6.2. Kalman filter parameters. The use of the Kalman filter in tracking algorithms is always a delicate issue. The first problem that needs to be solved is the estimation of the covariance matrix for the model noise and for the measurement noise: in our case the matrix Q has been tuned manually: for all the experiments described in (3) we choose:

$$Q = \begin{pmatrix} 10 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 1000 \end{pmatrix}$$

The structure of the matrix reflects the fact that the choice of a constant acceleration model does not resemble closely the dynamics of the pose parameters over long time intervals. As far as the covariance matrix R is concerned we used two values: $R = 0.6$ when we used a detailed model for the head, and $R = 1$ when we used either an ellipsoidal or a superellipsoidal model.

CHAPTER 3

Experimental results

In this chapter we describe some of the experimental results obtained using our tracking algorithm. These experiments are divided into two groups: in the first group synthesized images are used, whereas in the second group real video sequences are processed.

3.1. Synthesized sequences

The synthesized sequences allow us to carry out a quantitative analysis of the performance of the algorithm, since we know which are the ground truth values for the pose of the head. Those sequences are rendered using the OpenGL library and a detailed 3D model for the human head. The dimensions of the head have been set according to the RAMSIS anthropometric database [**7**] which provides the following average values for a US/Canadian head:

	head dimensions [mm]
head height	223
head width	159
head depth	194

Since a real human head turns out to be more rich in texture than the plain 3D model we rendered, we decided to add a texture to the head, so that the features can be tracked more easily by the feature tracking algorithm, resembling more closely what happens with real video sequences. The trajectory of the head is described by the sequence of matrices $G_{o2w}(n) = T(t(n)) \cdot R_z(\omega_z(n)) \cdot R_y(\omega_y(n)) \cdot R_x(\omega_x(n))$, $1 \leq n \leq N$, where:

$$T(t(n)) = \begin{pmatrix} 1 & 0 & 0 & t_x(n) \\ 0 & 1 & 0 & t_y(n) \\ 0 & 0 & 1 & t_z(n) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_x(\alpha(n)) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\omega_x(n) & -\sin\omega_x(n) & 0 \\ 0 & \sin\omega_x(n) & \cos\omega_x(n) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

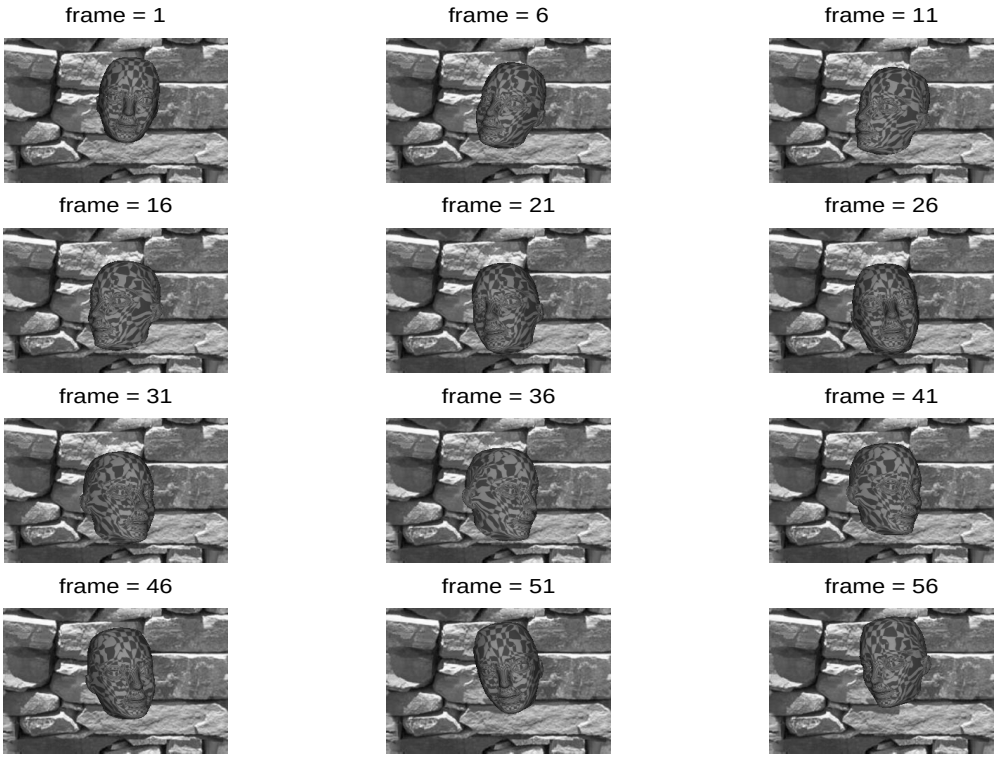

FIGURE 3.1.1. Some frames from a synthetic sequence.

$$R_y(\beta(n)) = \begin{pmatrix} \cos\omega_y(n) & 0 & \sin\omega_y(n) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\omega_y(n) & 0 & \cos\omega_y(n) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z(\gamma(n)) = \begin{pmatrix} \cos\omega_z(n) & -\sin\omega_z(n) & 0 & 0 \\ \sin\omega_z(n) & \cos\omega_z(n) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Particular care has also been taken for the camera model: in order to produce realistic sequences we choose to simulate a $\frac{1}{3}$inch format camera with a $4.8mm \times 3.6mm$ CCD sensor. The focal length has been set to $5.5mm$. For all the synthetic sequences we have:

$$G_{w2c} = T\left(\begin{pmatrix} 0 & 0 & d \end{pmatrix}^T\right) \cdot R_y\left(180^\circ\right)$$

Some frames of a typical synthesized sequence can be seen in figure (3.1.1).

3.1.1. The experiments. Three experiments have been carried out: the purpose of the first experiment is to study the behavior of the algorithm when

different models are used, the second experiment is devoted to the study of the performance of the algorithm in dependence of the number of features used in the tracking process and finally the last experiment is meant to study the influence of the initial fitting of the 3D model.

The trajectory of the head in each sequence is described by the equations in (3.1.1.1), (3.1.1.2) and (3.1.1.3).

3.1.1.1. *syn_ all01 synthetic sequence.*

- translation along x axis: $t_x(n) = -30 + 20 \cdot \sin\left(\frac{\pi}{N} \cdot n\right)$ $[mm]$
- translation along y axis: $t_y(n) = 30 - 50 \cdot \sin\left(\frac{\pi}{0.8 \cdot N} \cdot n\right)$ $[mm]$
- translation along z axis: $t_z(n) = 190 + 50 \cdot \sin\left(\frac{\pi}{N} \cdot n\right)$ $[mm]$
- roll angle: $\omega_x(n) = -15 \cdot \sin\left(\frac{\pi}{0.75 \cdot N} \cdot n\right)$ $[deg]$
- pitch angle: $\omega_y(n) = 40 \cdot \sin\left(\frac{\pi}{0.4 \cdot N} \cdot n\right)$ $[deg]$
- yaw angle: $\omega_z(n) = +10 \cdot \sin\left(\frac{\pi}{0.25 \cdot N} \cdot n + \frac{\pi}{5}\right)$ $[deg]$

3.1.1.2. *syn_ all02 synthetic sequence.*

- translation along x axis: $t_x(n) = -10 + 10 \cdot \sin\left(\frac{\pi}{N} \cdot n\right)$ $[mm]$
- translation along y axis: $t_y(n) = 15 - 15 \cdot \sin\left(\frac{\pi}{0.8 \cdot N} \cdot n\right)$ $[mm]$
- translation along z axis: $t_z(n) = 180 + 20 \cdot \sin\left(\frac{\pi}{N} \cdot n\right)$ $[mm]$
- roll angle: $\omega_x(n) = -7 \cdot \sin\left(\frac{\pi}{0.75 \cdot N} \cdot n\right)$ $[deg]$
- pitch angle: $\omega_y(n) = 20 \cdot \sin\left(\frac{\pi}{0.4 \cdot N} \cdot n\right)$ $[deg]$
- yaw angle: $\omega_z(n) = +5 \cdot \sin\left(\frac{\pi}{0.25 \cdot N} \cdot n + \frac{\pi}{5}\right)$ $[deg]$

3.1.1.3. *syn_ all03 synthetic sequence.*

- translation along x axis:
  $t_x(n) = 10 \cdot \left[\sin\left(\frac{\pi}{N} \cdot n\right) + \cos\left(\frac{\pi}{0.45 \cdot N} \cdot n\right) + \sin\left(\frac{\pi}{1.43 \cdot N} \cdot n + \frac{\pi}{3}\right)\right]$ $[mm]$
- translation along y axis:
  $t_y(n) = -20 + 20 \cdot \left[\sin\left(\frac{\pi}{N} \cdot n\right) + \left(\cos\left(\frac{\pi}{0.56 \cdot N} \cdot n\right)\right)^2 + \sin\left(\frac{\pi}{1.43 \cdot N} \cdot n + \frac{\pi}{5}\right)\right]$ $[mm]$
- translation along z axis:
  $t_z(n) = 20 \cdot \left[\sin\left(\frac{\pi}{N} \cdot n\right) + \left(\cos\left(\frac{\pi}{0.56 \cdot N} \cdot n\right)\right)^2 + \sin\left(\frac{\pi}{1.43 \cdot N} \cdot n + \frac{\pi}{5}\right)\right]$ $[mm]$
- roll angle: $\omega_x(n) = 15 \cdot \sin\left(\frac{\pi}{3.33 \cdot N} \cdot n\right) + \left(\frac{n}{5}\right)^{\frac{1}{3}}$ $[deg]$
- pitch angle: $\omega_y(n) = 15 \cdot \sin\left(\frac{\pi}{3.33 \cdot N} \cdot n\right) - \left(\frac{n}{5}\right)^{\frac{1}{3}}$ $[deg]$
- yaw angle: $\omega_z(n) = +10 \cdot \sin\left(\frac{\pi}{0.29 \cdot N} \cdot n + \frac{\pi}{5}\right) - \left(\frac{n-N}{20}\right)^2$ $[deg]$

The previous sequences test quite severely the performance of the tracking algorithm, since large variations of the pose parameters may occur over a small number of frames: in sequence (3.1.1.1), which is the test sequence where the largest motions are involved, the roll angle varies of 80° in about 25 frames, and the head centroid of about $7cm$. In the last two sequences (i.e. (3.1.1.2) and

(3.1.1.3)) the parameter variation are smaller, and the trajectory described resembles more closely a trajectory that may be described by the head in tasks such gesture recognition or vision driven human computer interfaces.

3.1.2. Experiment one: the model. For all the three sequences composed by 60 frames 60 features have been used. For each sequence there are three graphics for each pose parameter: one graphic where the ground truth values (ϕ_i) are plotted against the values recovered by the tracker ($\widehat{\phi}_i$), another graphic where the error $e_{\phi_i} = \phi_i - \widehat{\phi}_i$ is plotted and finally a graphic with the root mean square error $e_{rms,\phi_i} = \frac{1}{n}\sum_{j=1}^{n} e_{\phi_i}^2$.

NOTE 3.1.1. In the figures that contain the results of the experiments (from (3.1.2) to (3.1.4)) red graphs correspond to the ellipsoidal model, green graphs to the superellipsoidal model and blue graphs to the detailed head model. Where present, ground truth values are plotted in black.

From the results of the experiments it becomes clear that the performance of the algorithm is conditioned by two aspects: the speed and the amplitude of the movements of the head. The first aspect is related to the capability of the feature tracking algorithm to deal with large frame to frame displacements of the features, whereas the second aspect is related to the model disagreement that may become relevant for particular poses of the head which strongly differ from the initial pose.

A good index to measure the performance of the algorithm is given by the root mean square error: using this quantity it is possible to estimate the cumulative error introduced during the tracking process. From the experiments it is evident that the ellipsoidal model introduces the greatest cumulative tracking error (and this can be explained by the fact that for large motions the model disagreement using an ellipsoidal model for the head becomes quite relevant), whereas for the sequences (3.1.1.2) and (3.1.1.3) the best performance is achieved using a detailed head model (the maximum error for the translation parameters is about $e_{t,max} \cong 0.5cm$, and for the angle parameters $e_{\omega,max} \cong 4°$). As far as the sequence (3.1.1.1) is concerned, good results have been obtained using a superellipsoidal model for the head (in this case the maximum error for the translation parameters is about $e_{t,max} \cong 3.5cm$, and for the angle parameters $e_{\omega,max} \cong 7°$). Form the inspection of the graphics, it is also evident that the estimation of those parameters whose variation causes a motion of the feature points along a direction that is parallel to the optical axis is the most difficult. This can be explained observing that feature points that undergo relevant translation along a direction parallel to the optical axis describe short trajectories on the image plane, whose estimation by the tracking algorithm might be not so much accurate (even when the displacement

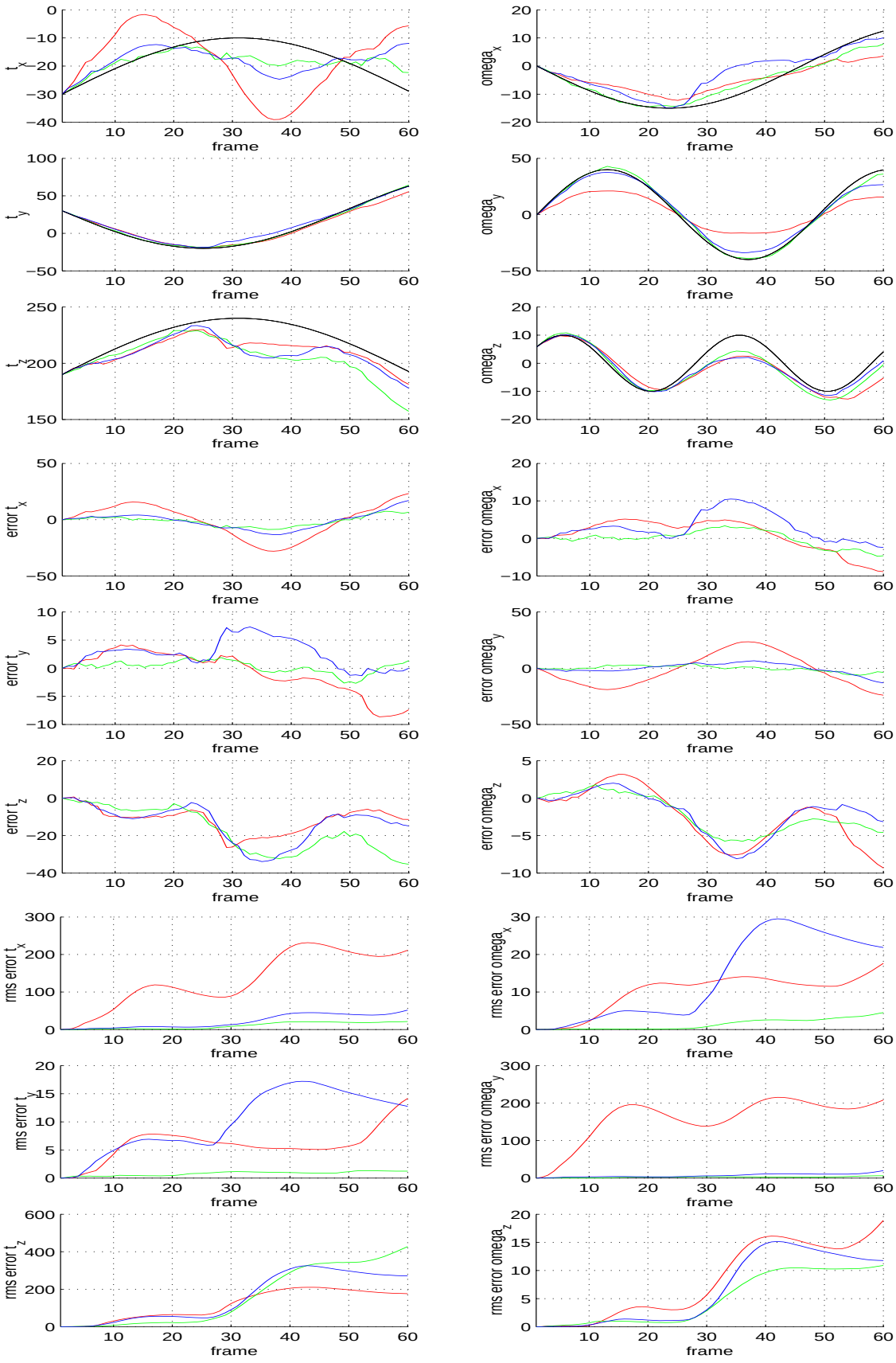

FIGURE 3.1.2. Sequence syn_all01: pose parameters and corresponding errors.

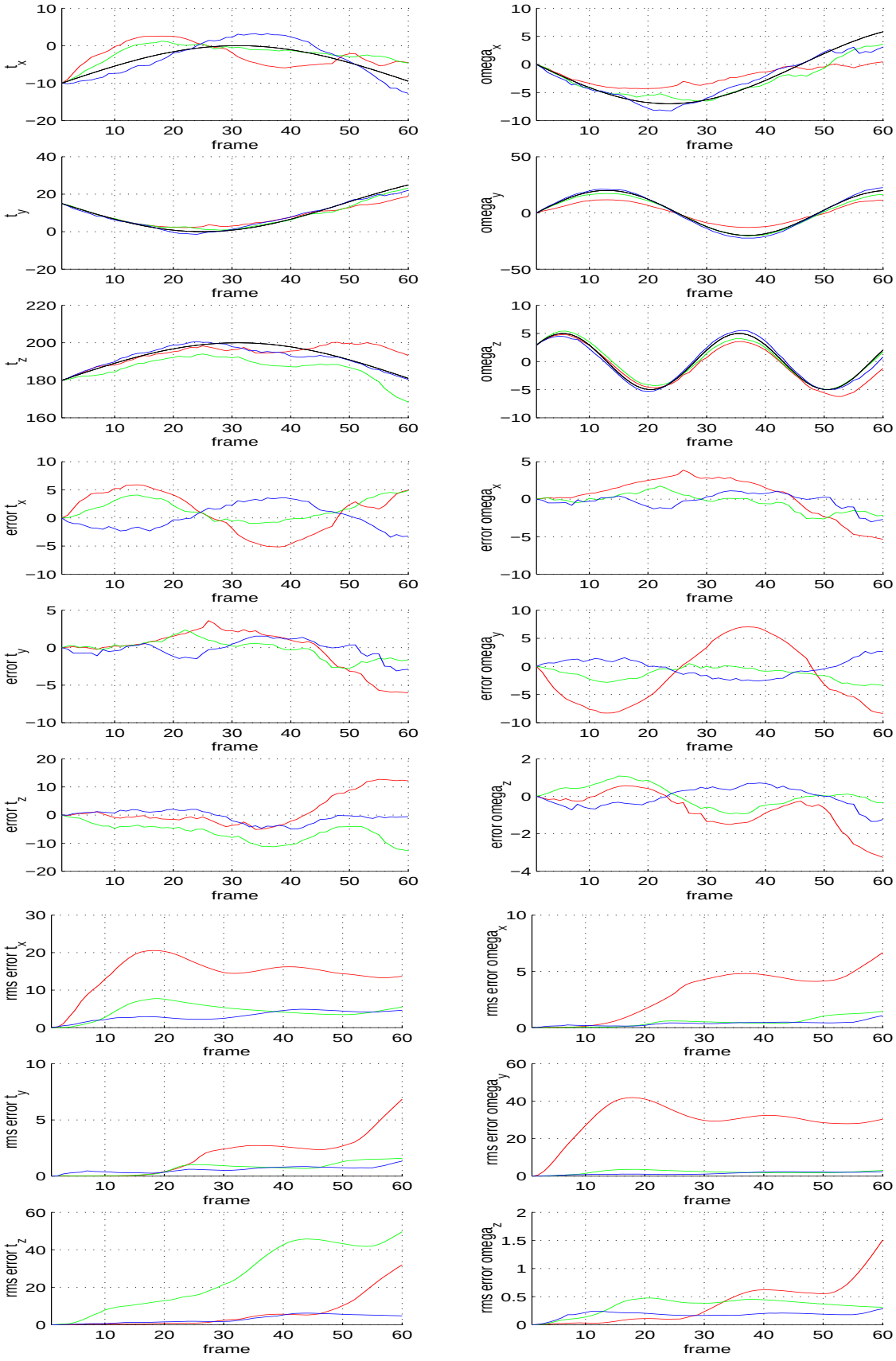

FIGURE 3.1.3. Sequence syn_all02: pose parameters and corresponding errors.

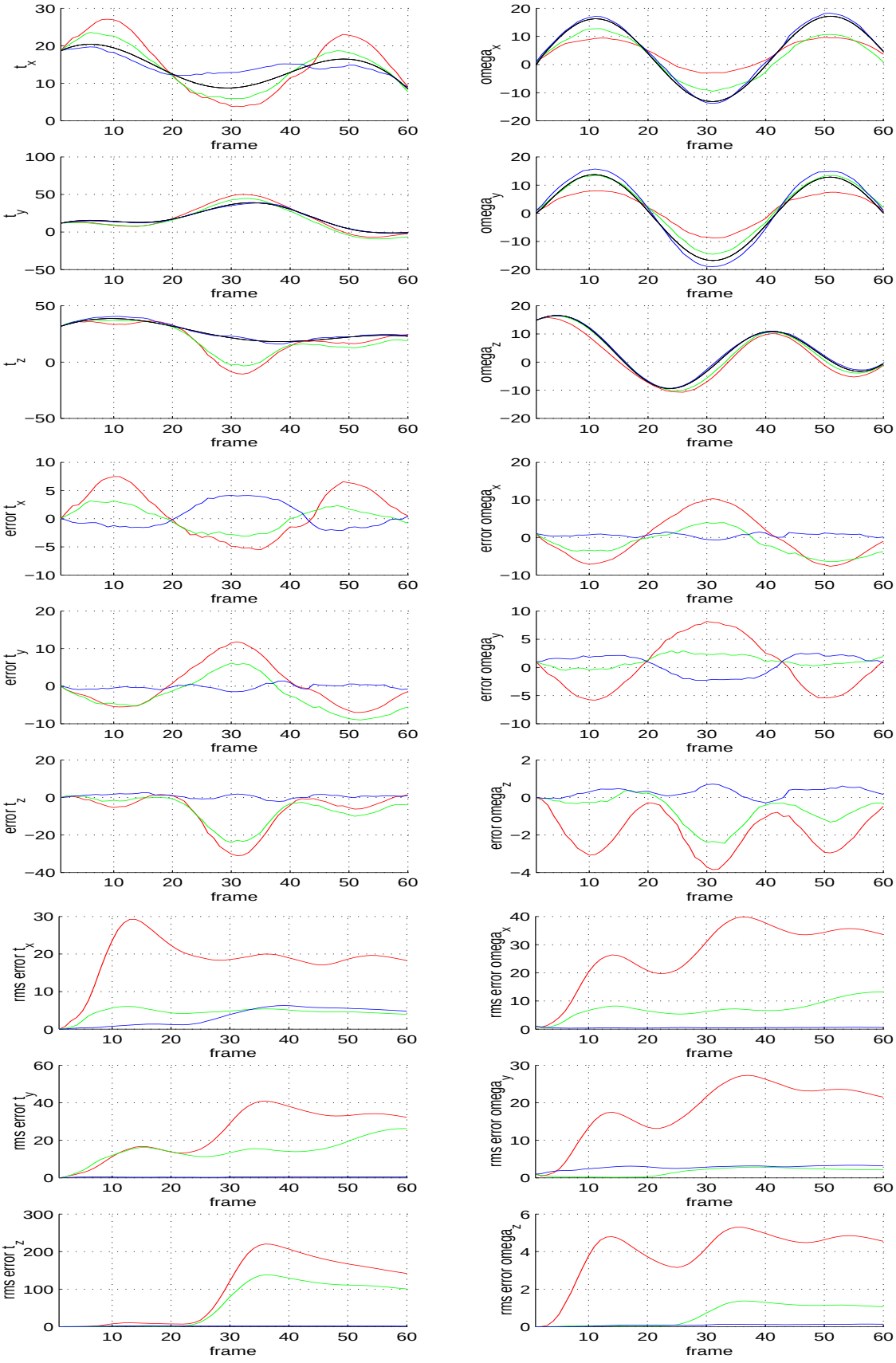

FIGURE 3.1.4. Sequence syn_all03: pose parameters and corresponding errors.

vector is estimated in a Newton Raphson fashion which allows to obtain subpixel accuracy). Another fact that may confuse the tracking algorithm is related to those movements of the head (and consequently of the features point on the image plane) that, because of the noise superimposed to the measurements, cannot be related univocally to a change of some components of the parameter vector ϕ. This may happen for example for a small translation of the head parallel to the x axis of Σ_o or for a small rotation along the y axis: in both cases the trajectory described by the features points on the image plane is similar. Finally we can notice that in the sequences (3.1.1.2) and (3.1.1.3), where the variation of the pose parameters is not so large as in (3.1.1.1), the cumulative tracking error remains almost constant: the important consequence of this fact is that the head can be tracked accurately over many frames.

3.1.3. Experiment two: the number of features. In this experiment we compared the tracking results obtained using different number of features (see (2.6.1)) for the same video sequence. The meaning of the graphics has been described in (3.1.2).

NOTE 3.1.2. In the figures that contain the results of the experiments red graphs correspond to the results obtained using 20 features, green graphs to 40 features and finally blue graphs to 60 features. Where present, ground truth values are plotted in black.

From the experiments we can notice that the effect of the number of the features used for the tracking procedure is not so relevant when an ellipsoidal model for the head is used. Instead when the superellipsoidal model and the detailed model are used, we can say that the greater is the number of the features used the better is the performance of the algorithm (this is particularly clear moving from 20 features to 40 features when processing the sequence (3.1.1.1)). This fact can be easily understood observing that a large number of feature points used in the least square minimization procedure contributes to reduce the effects both of the noise introduced by the feature tracking algorithm and of the errors due to the model disagreement. Anyway there is also another issue that affects the performance of the algorithm when the number of the features is low: the distribution of the feature points inside the apparent contour γ may be not homogeneous[1]. When the features are located in a small subregion inside γ, the algorithm may be fooled, trying to explain the trajectories of the feature points on the image with a trajectory of the head that is deeply different from the real one.

The dependence of the performance of the tracking algorithm from the number of the features is a critical issue when we want to consider the possibility of a real time implementation, since all the basic routines that compose the global algorithm have a complexity that is linear in the number of the features. Therefore there is a trade-off between the accuracy of the tracker and the number of frames per second that can be processed.

[1]When the number of features is 20, the distance constraint described in (2.6.1) is not enough to ensure the homogeneous distribution of those points inside γ.

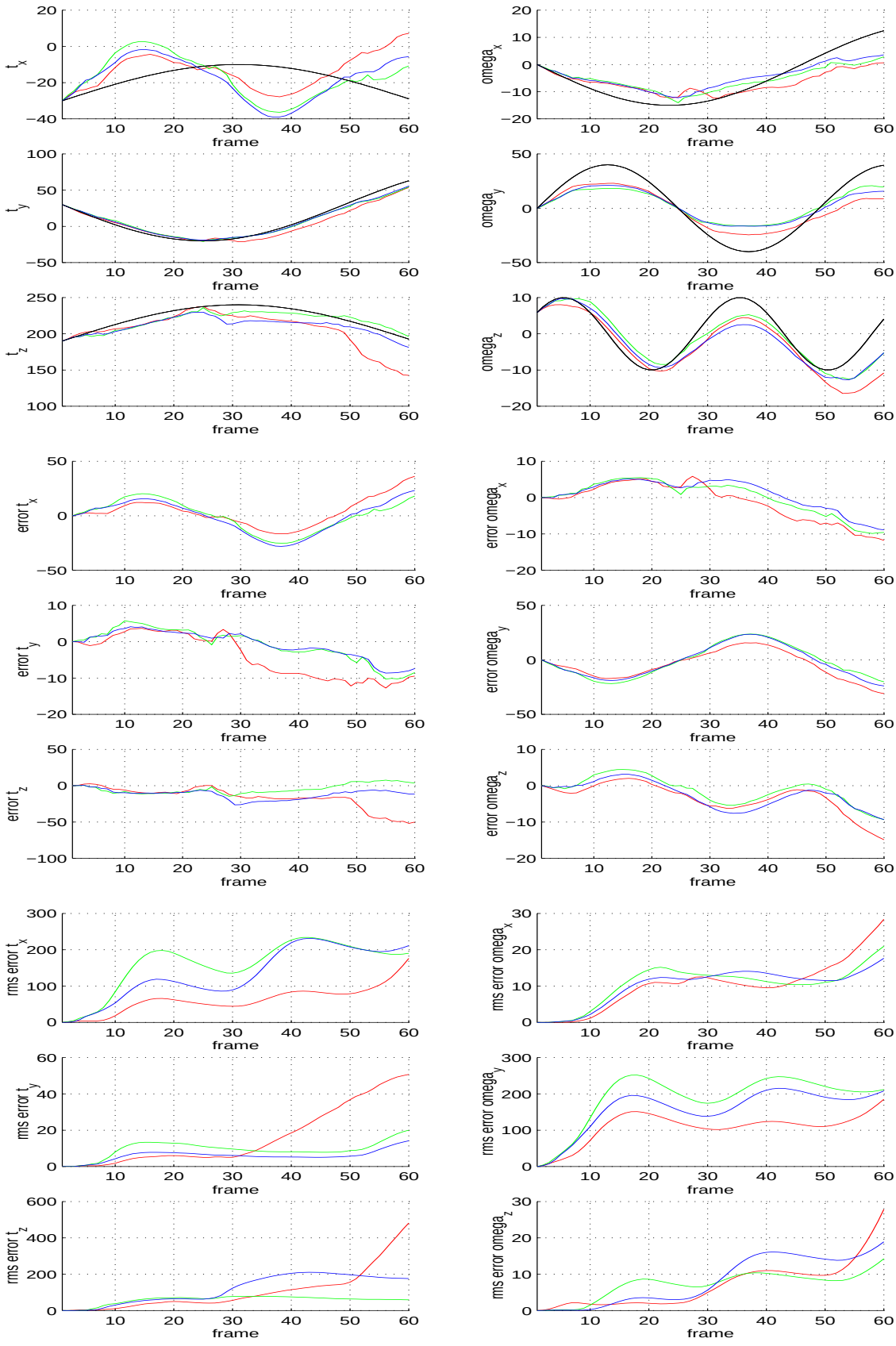

FIGURE 3.1.5. Sequence syn_all01: pose parameters and corresponding errors with ellipsoidal head model.

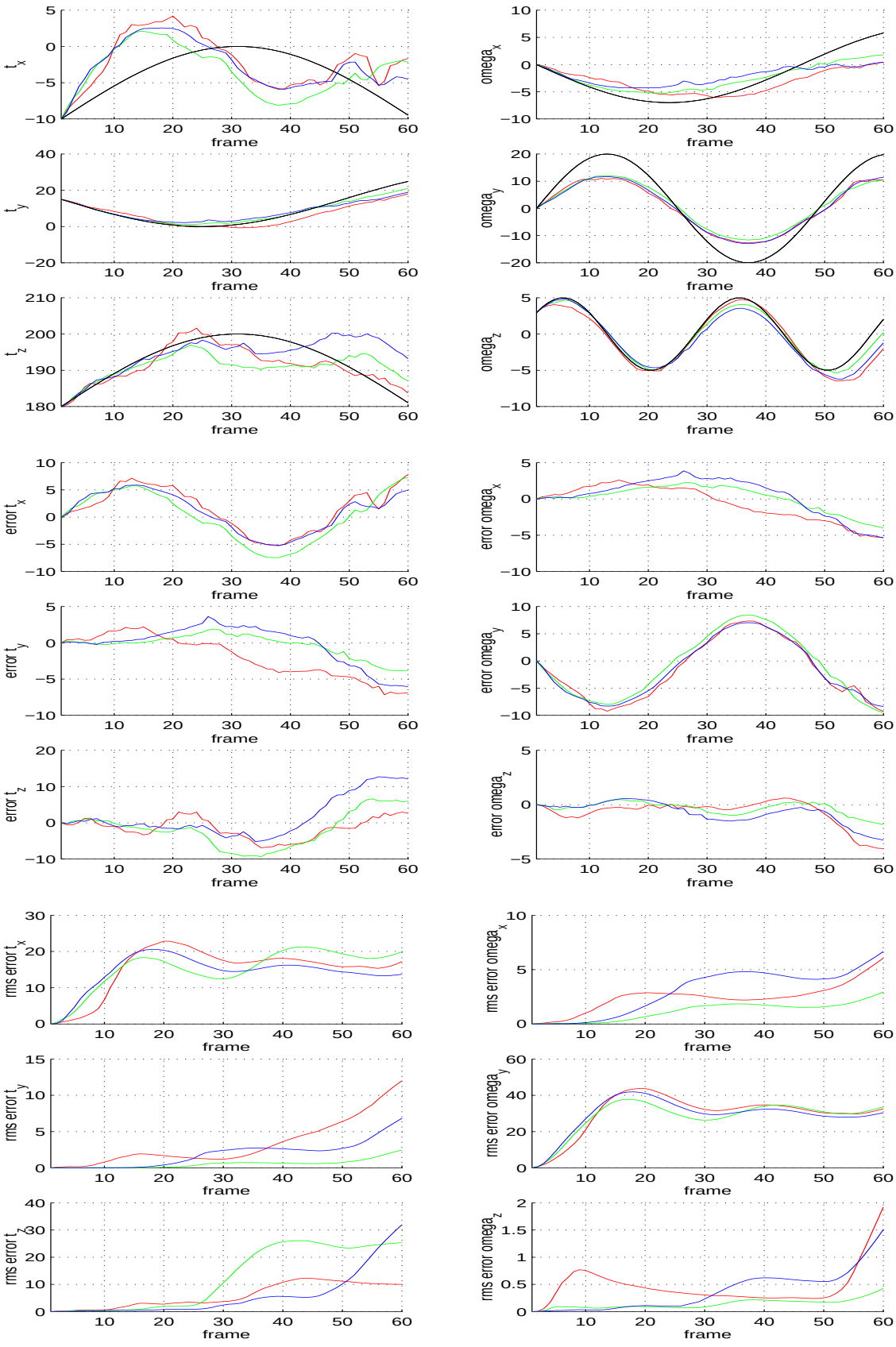

FIGURE 3.1.6. Sequence syn_all02: pose parameters and corresponding errors with ellipsoidal head model.

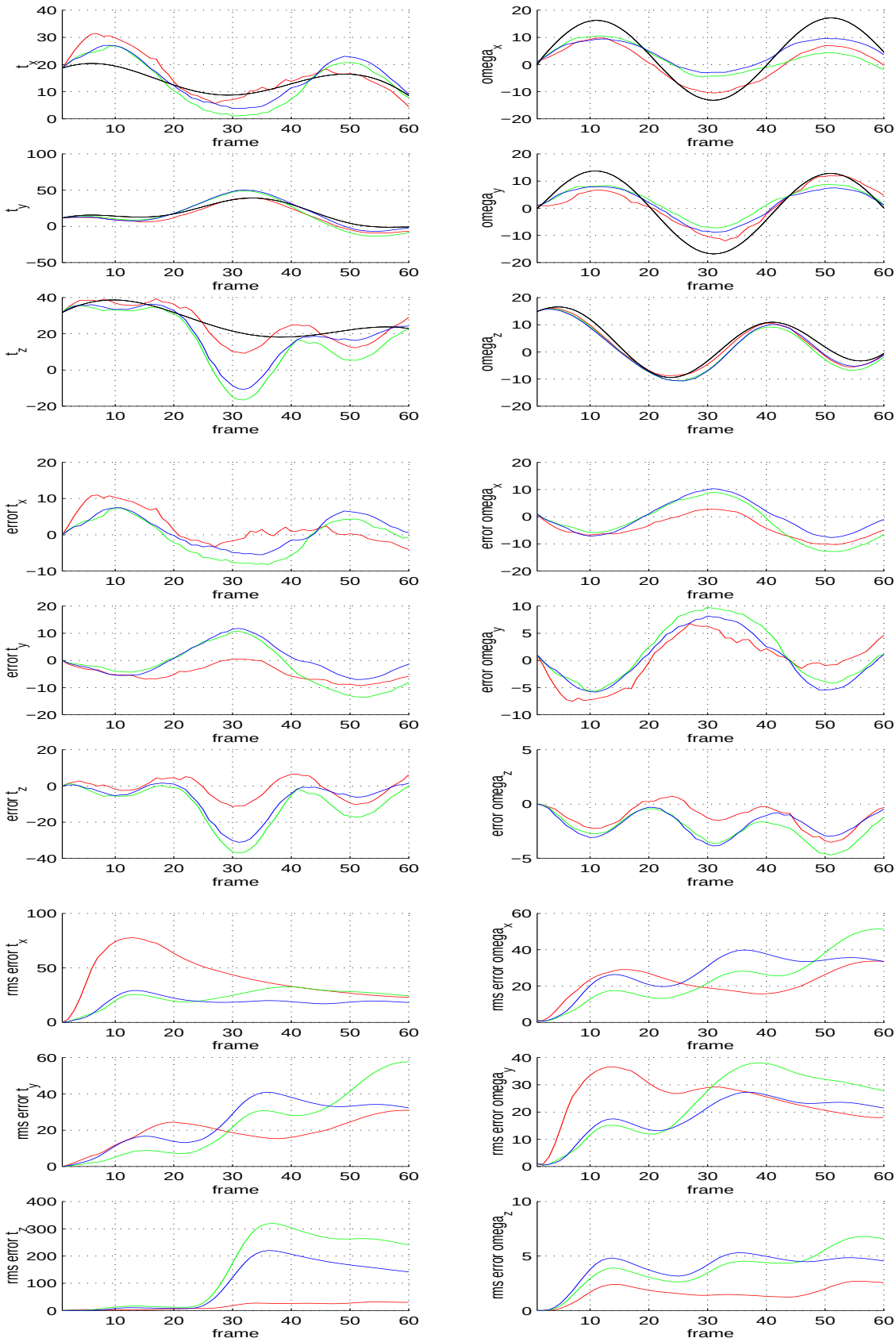

FIGURE 3.1.7. Sequence syn_all03: pose parameters and corresponding errors with ellipsoidal head model.

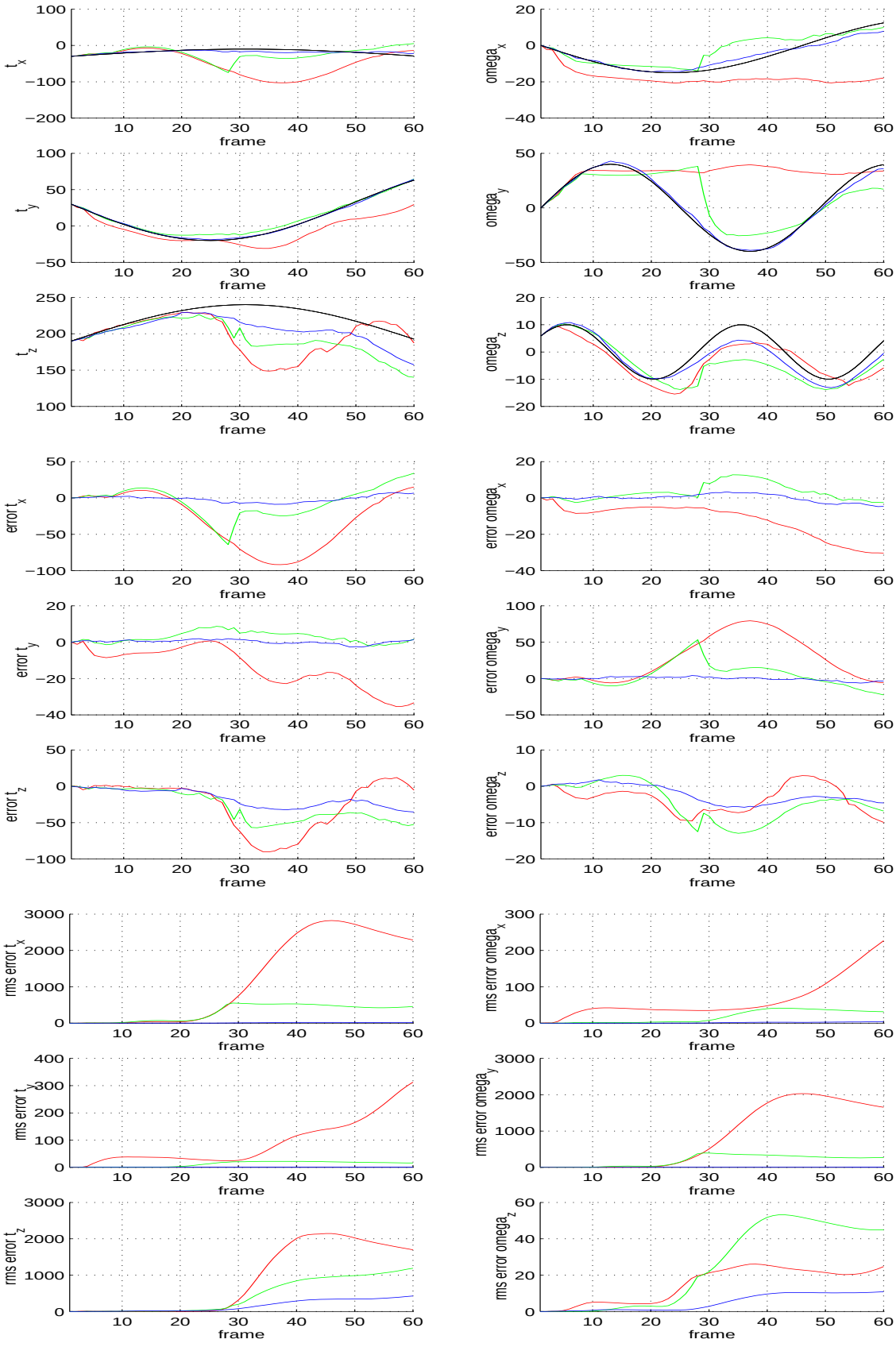

FIGURE 3.1.8. Sequence syn_all01: pose parameters and corresponding errors with superellipsoidal head model.

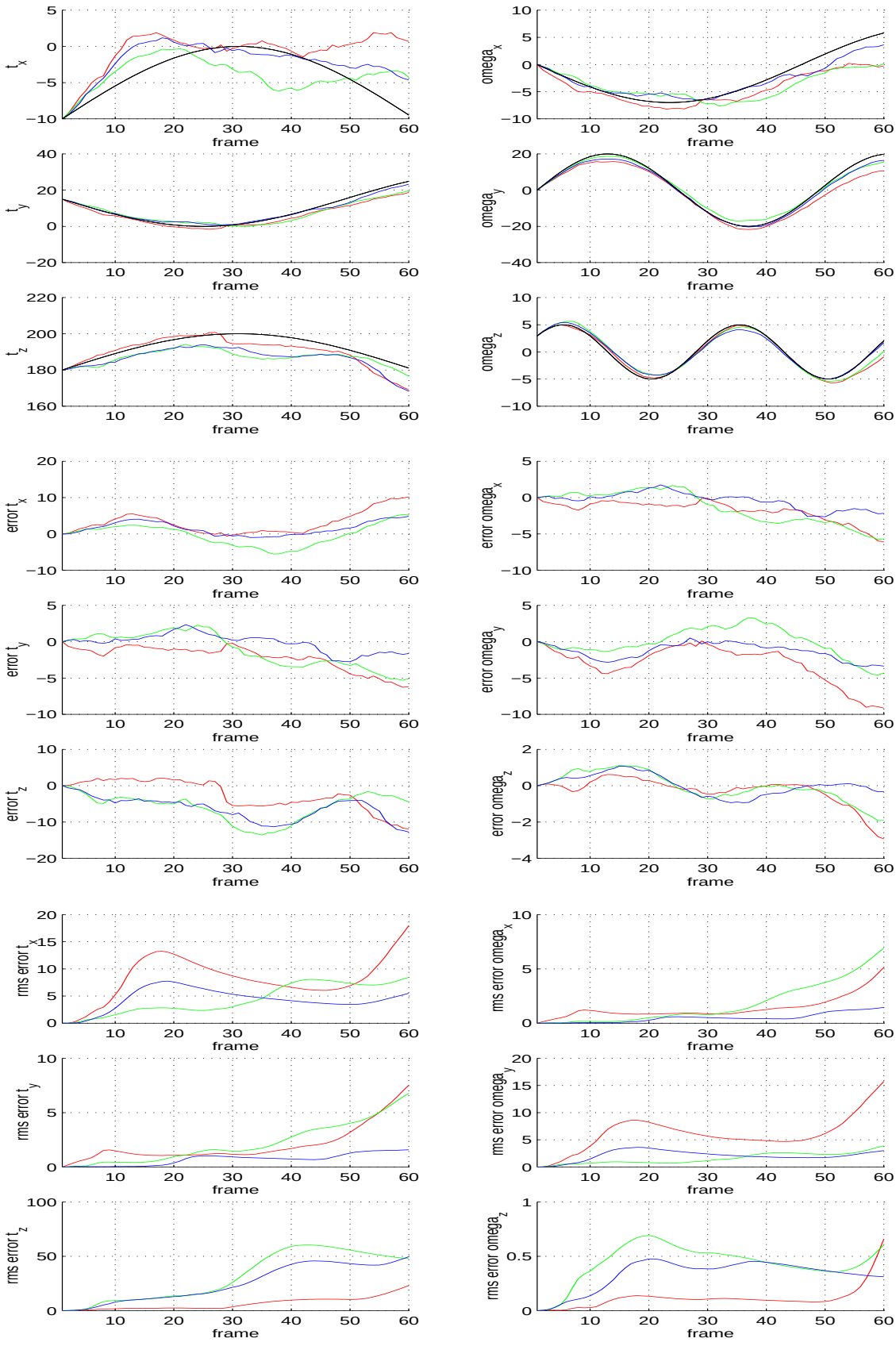

FIGURE 3.1.9. Sequence syn_all02: pose parameters and corresponding errors with superellipsoidal head model.

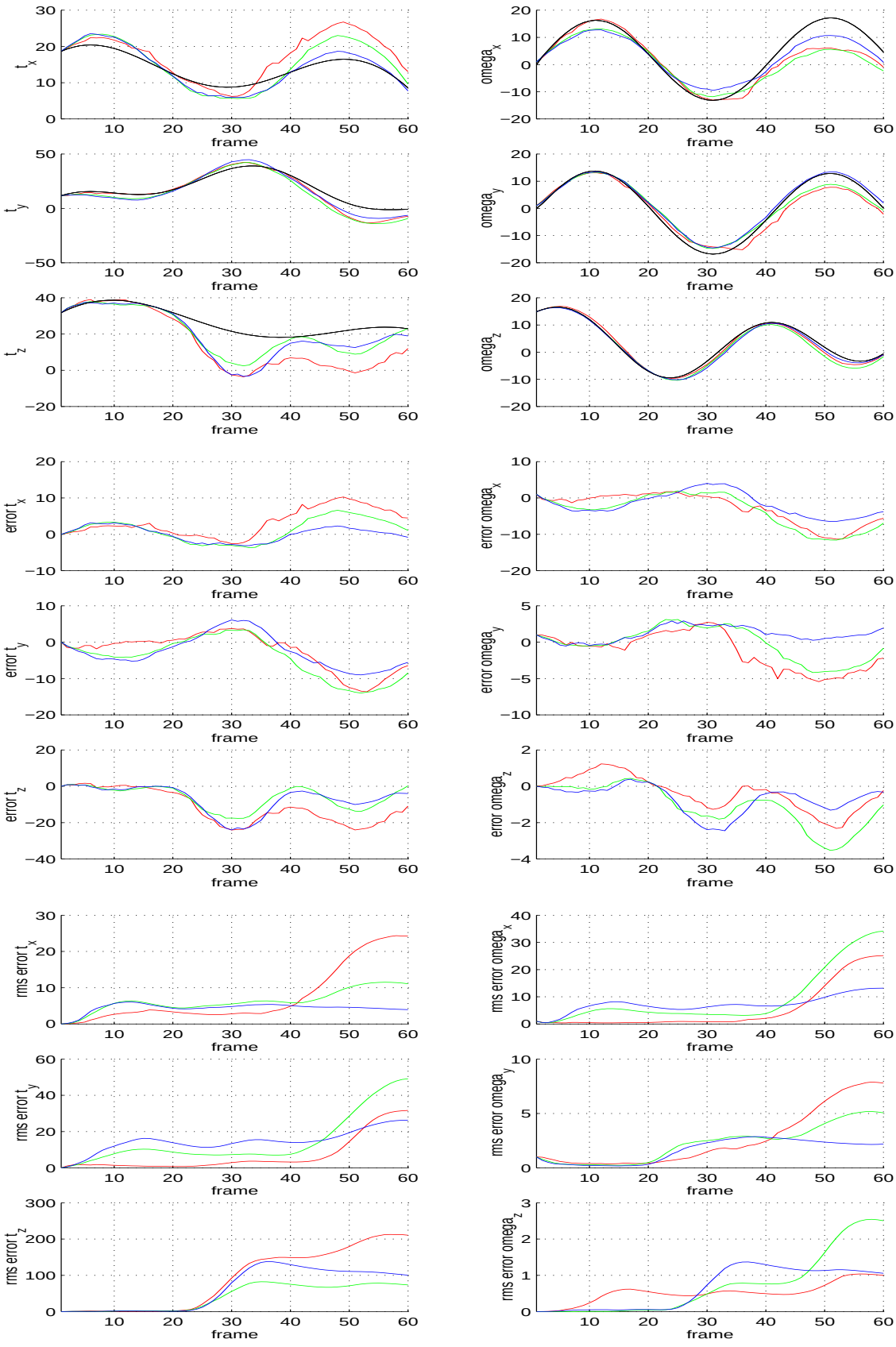

FIGURE 3.1.10. Sequence syn_all03: pose parameters and corresponding errors with superellipsoidal head model.

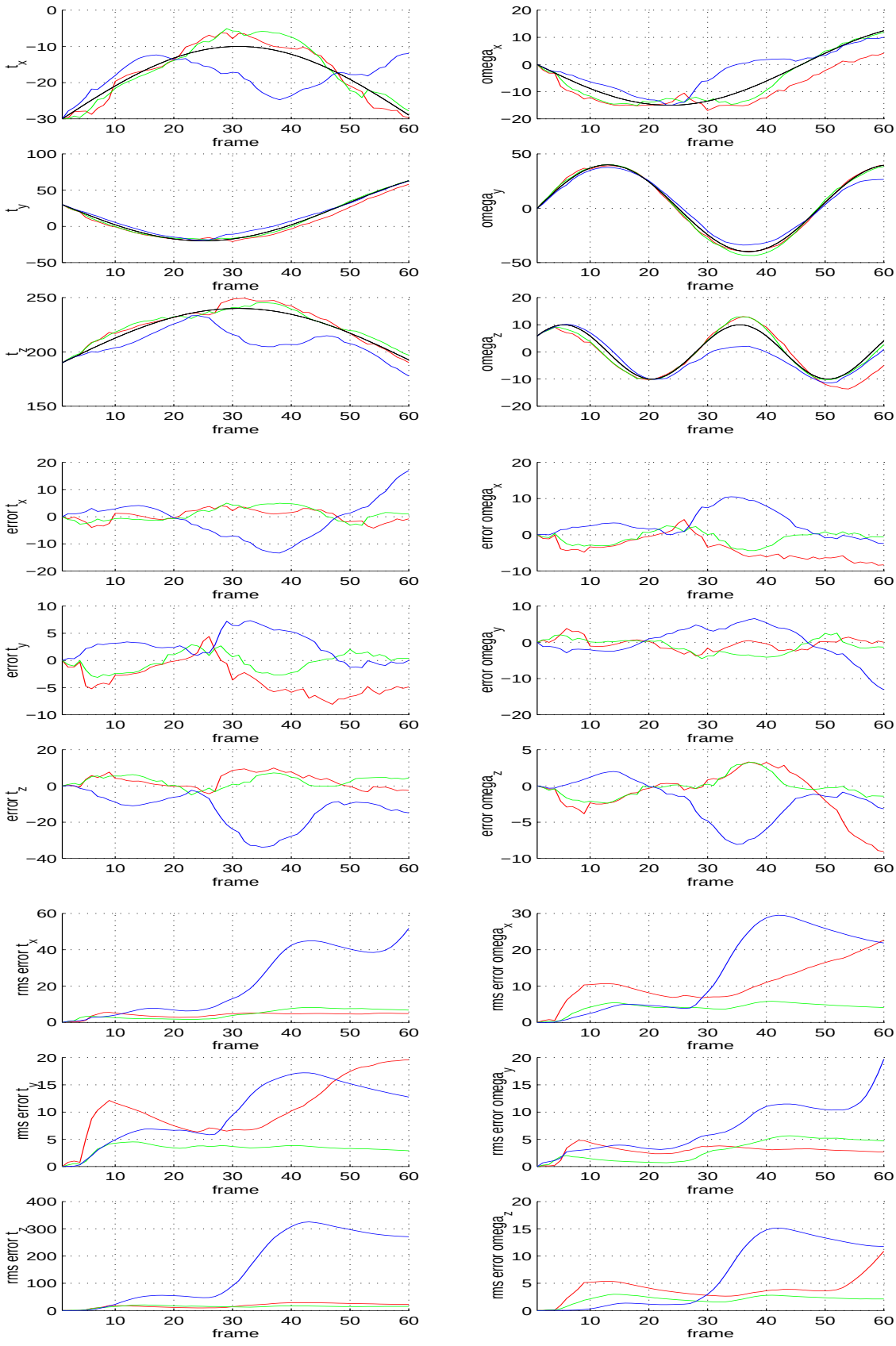

FIGURE 3.1.11. Sequence syn_all01: pose parameters and corresponding errors with detailed head model.

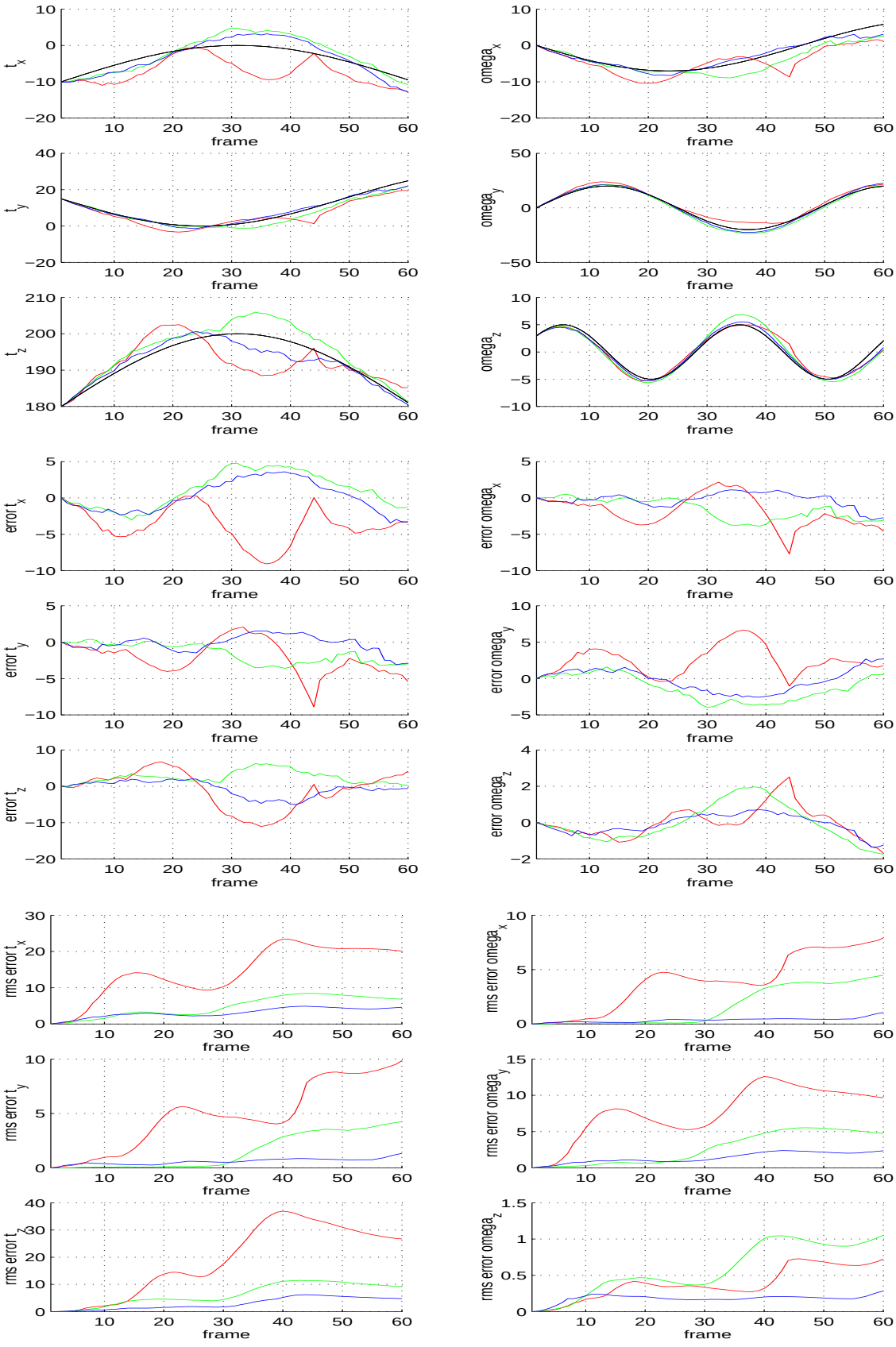

FIGURE 3.1.12. Sequence syn_all02: pose parameters and corresponding errors with detailed head model.

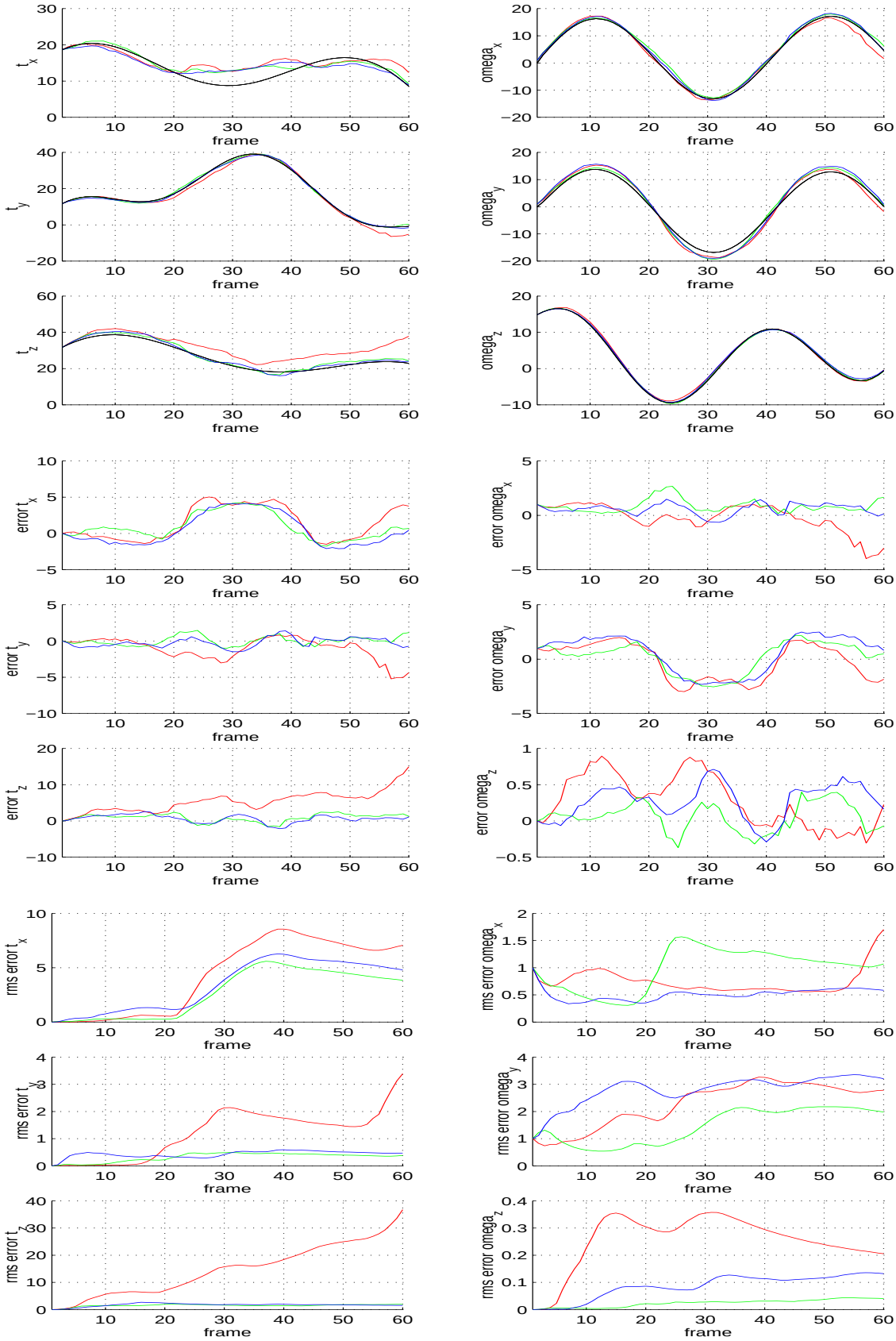

FIGURE 3.1.13. Sequence syn_all03: pose parameters and corresponding errors with detailed head model.

3.1.4. Experiment three: the initial fitting. In this experiment we are interested in the evaluation of the tracker's performances when the initial fitting of the model does not correspond to the ground truth values. To achieve this task the experiments described in (3.1.2) have been repeated, but the initial pose of the model has been perturbed in the following way:

- $t_x(1) \leftarrow t_x(1) + 5\ [mm]$
- $t_y(1) \leftarrow t_y(1) + 5\ [mm]$
- $t_z(1) \leftarrow t_z(1) + 15\ [mm]$
- $\omega_x(1) \leftarrow \omega_x(1) + 3^\circ$
- $\omega_y(1) \leftarrow \omega_y(1) + 3^\circ$
- $\omega_z(1) \leftarrow \omega_z(1) + 3^\circ$

NOTE 3.1.3. In the figures that contain the results of the experiments red graphs correspond to the ellipsoidal model, green graphs to the superellipsoidal model and blue graphs to the detailed head model. Where present, ground truth values are plotted in black. In order to make the comparison between the different plots more meaningful, the values of the initial perturbations have been subtracted to the values retrieved by the tracker.

The major conclusion that can be drawn from the experiments is that the perturbation of the initial pose parameters does not seem to affect much the performance of the algorithm (except for a fixed offset in the pose). This is true also when we use a detailed head model, and this fact is interesting if compared to the considerations present in **[10]**, where, using an optimization framework based on the matching of the optical flow (see (1.3.1)), the authors state:

> Too complex a model, such as an actual head, would require a very exact initial fit. If a detailed model were not fit accurately, the detailed features of the model could cause more harm than good.

The results of this experiment suggest the possibility to carry out the initial fitting of the 3D model on the image plane using some automatic procedure, using a system that is able to find the location of the head and the position of some features like eyes, nose and mouth that will allow to recover both the model parameters and the pose of the head at the initial frame.

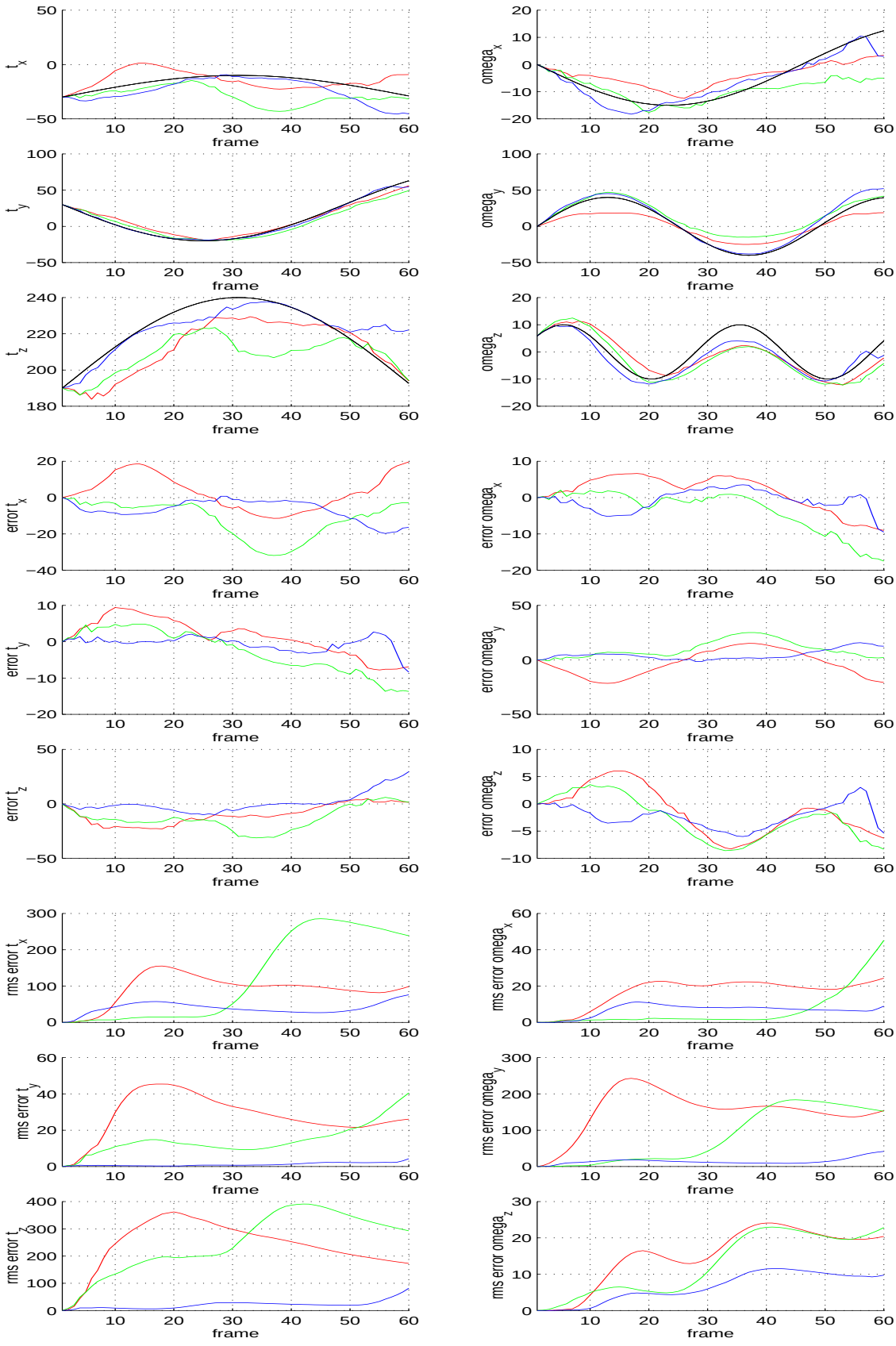

FIGURE 3.1.14. Sequence syn_all01: pose parameters and corresponding errors.

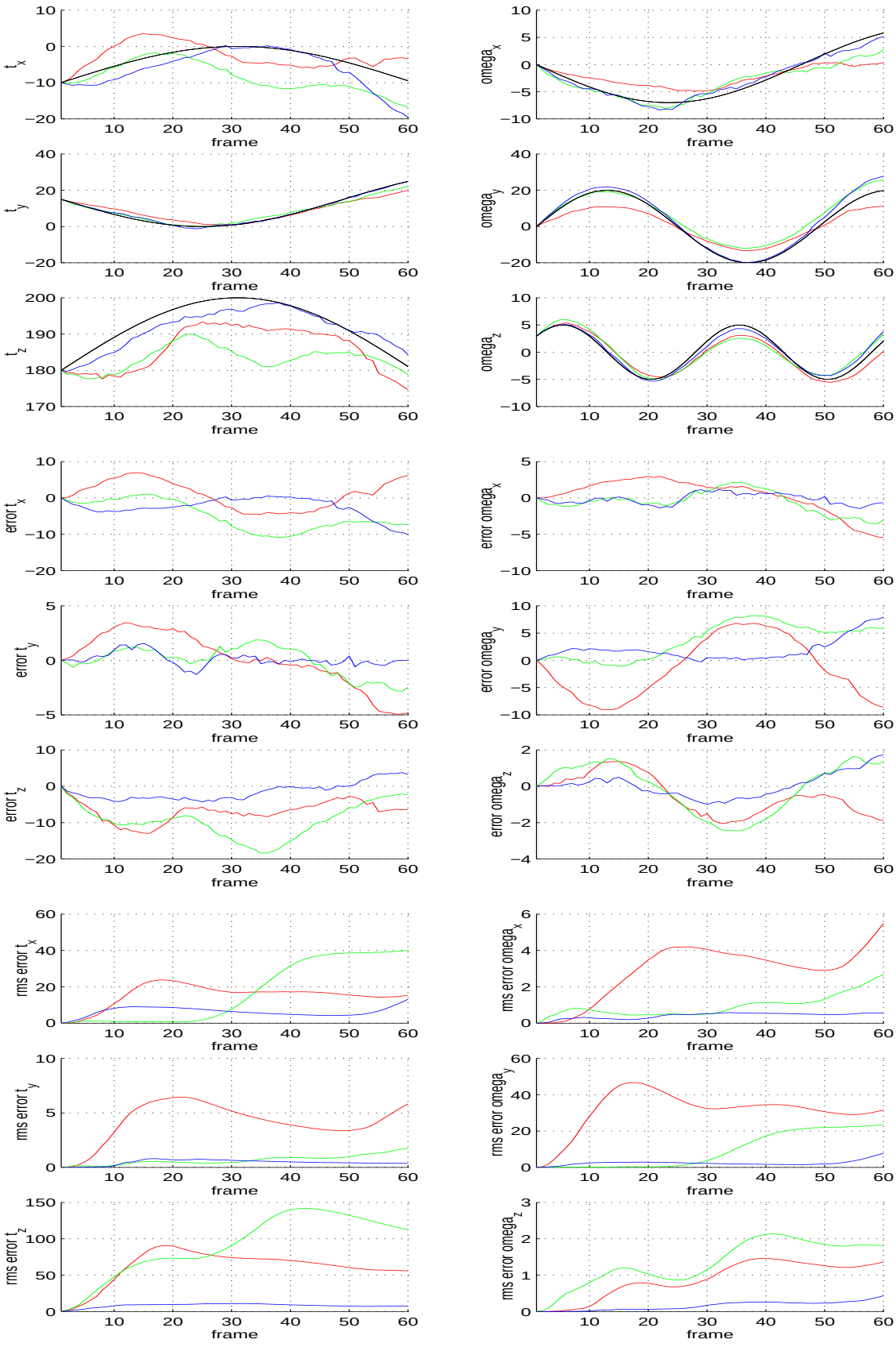

FIGURE 3.1.15. Sequence syn_all02: pose parameters and corresponding errors.

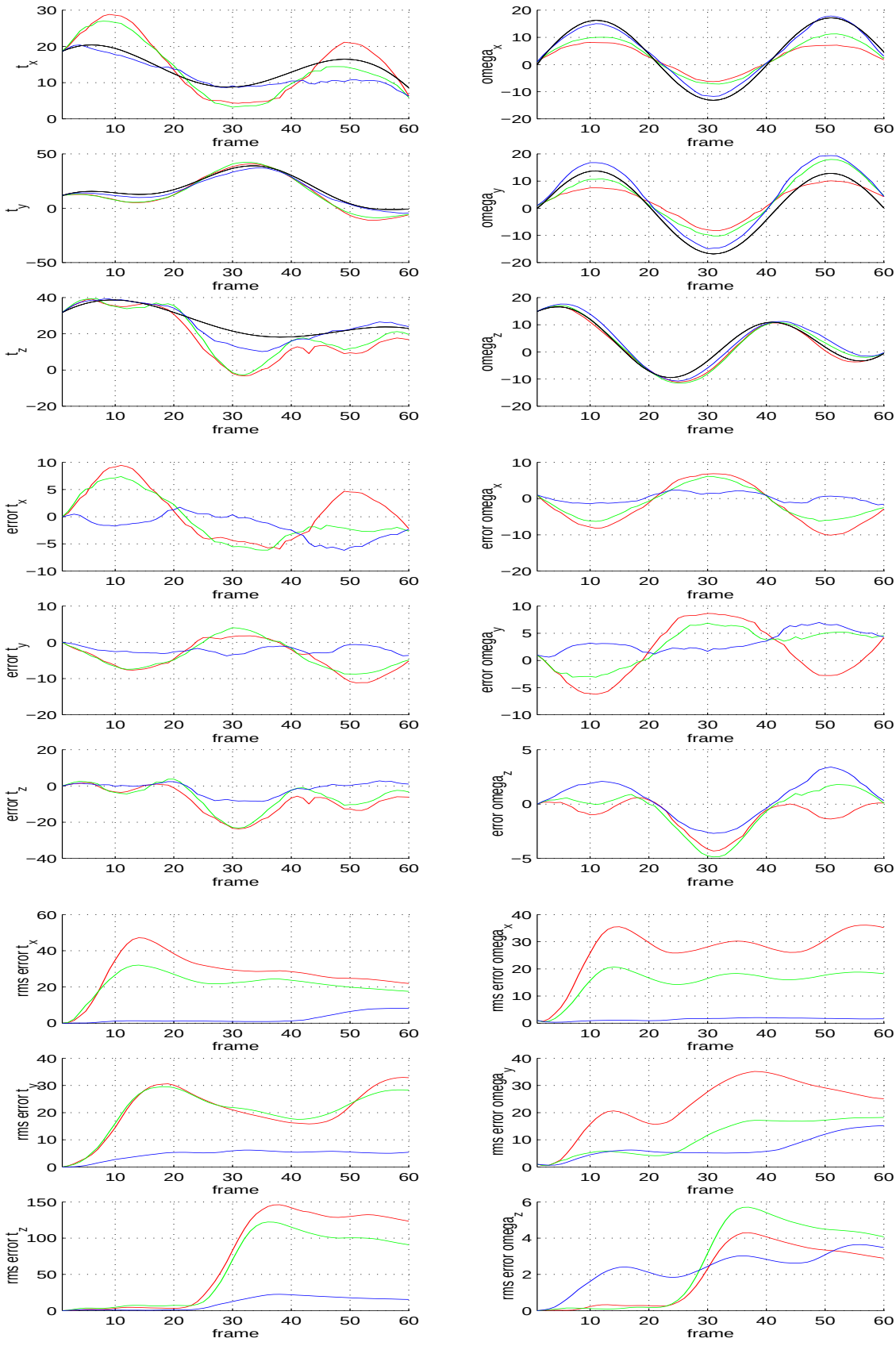

FIGURE 3.1.16. Sequence syn_all03: pose parameters and corresponding errors.

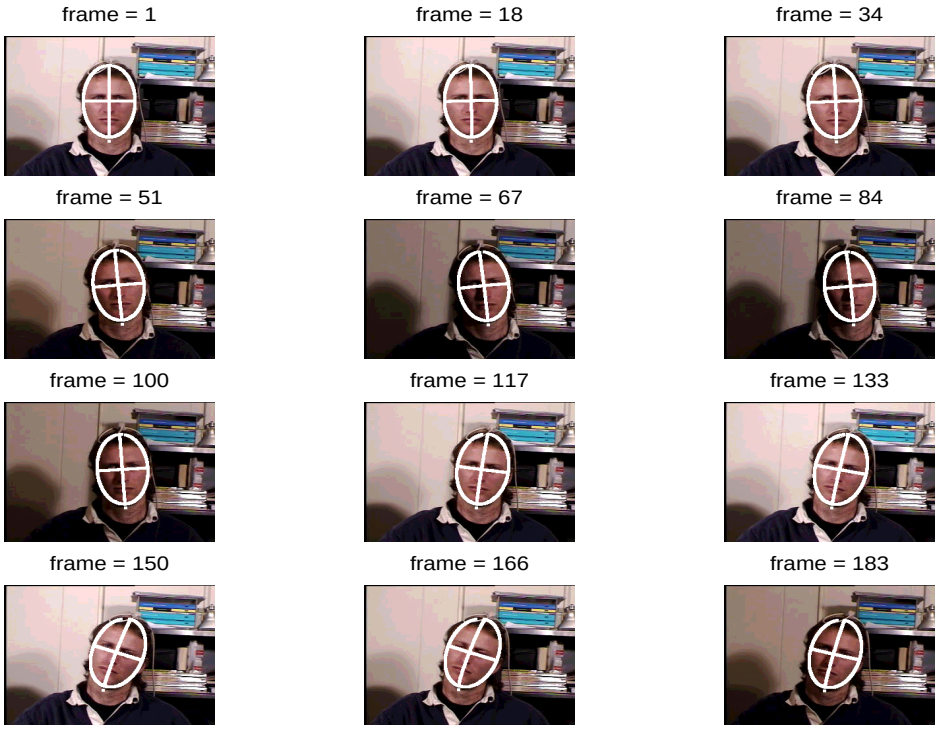

FIGURE 3.2.1. Tracking of a real video sequence using an ellipsoidal model

3.2. Real Sequences

In this experiment we test the performance of the algorithm on a real sequence under strongly varying lighting conditions: all the three models for the head have been used and the number of the features tracked is 60. The sequence is composed by 198 frames. Figures (3.2.1) through (3.2.3) show the different models superimposed to the head, whereas figure (3.2.4) shows the retrieved pose parameters.

Unfortunately, as already said, we do not have the ground truth values for this video sequence and therefore it is not possible to give a quantitative evaluation of the performance of the algorithm. Nevertheless we can try to interpret qualitatively the results in figure (3.2.4). First of all we notice that the trajectory retrieved by the tracking algorithm using the superellipsoidal model or the detailed head model is very similar, except for the translation parameter t_z and the angle parameter ω_y (note that in both cases the variation of these parameters causes the feature points to move along a direction parallel to the optical axis). Secondly we notice from the movie obtained superimposing the model to the image of the head that the model seem to reproduce with good accuracy the

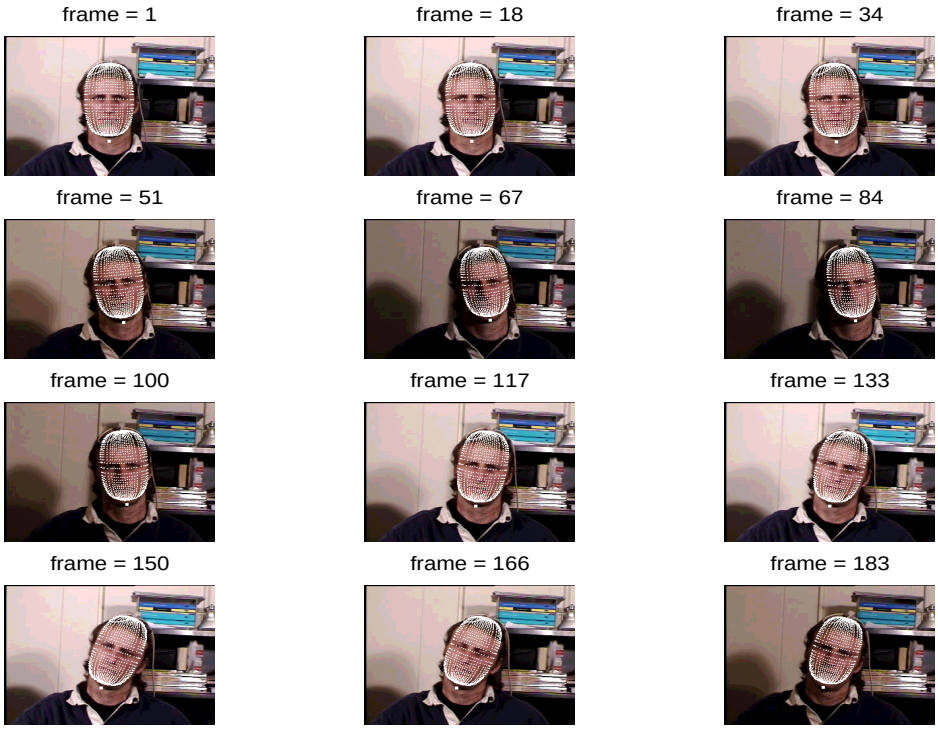

FIGURE 3.2.2. Tracking of a real video sequence using a superellipsoidal model

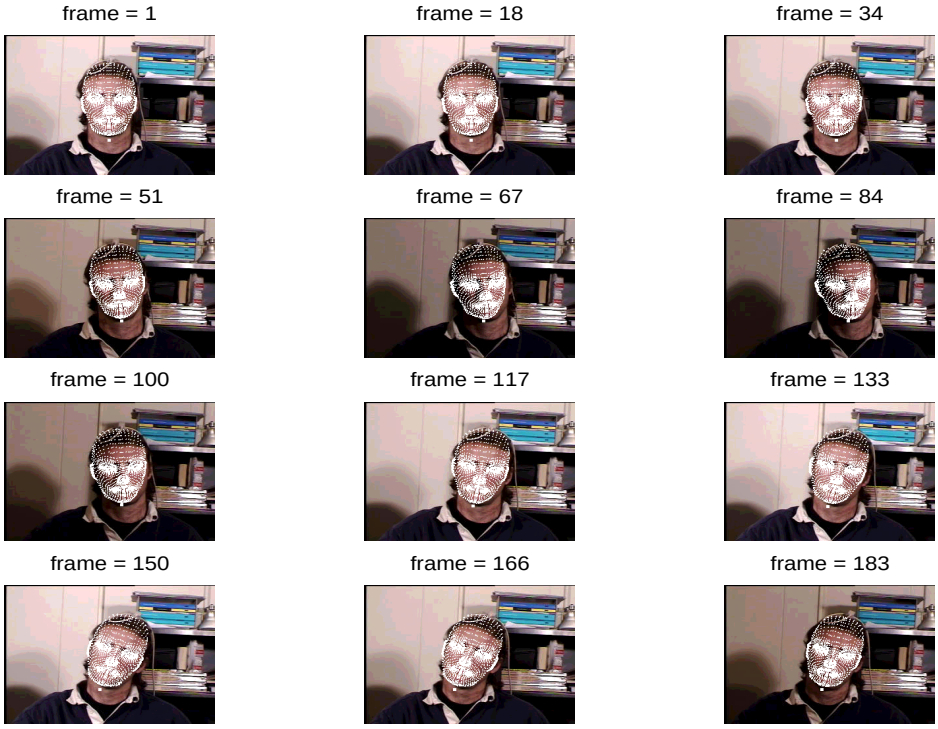

FIGURE 3.2.3. Tracking of a real video sequence using a detailed head model

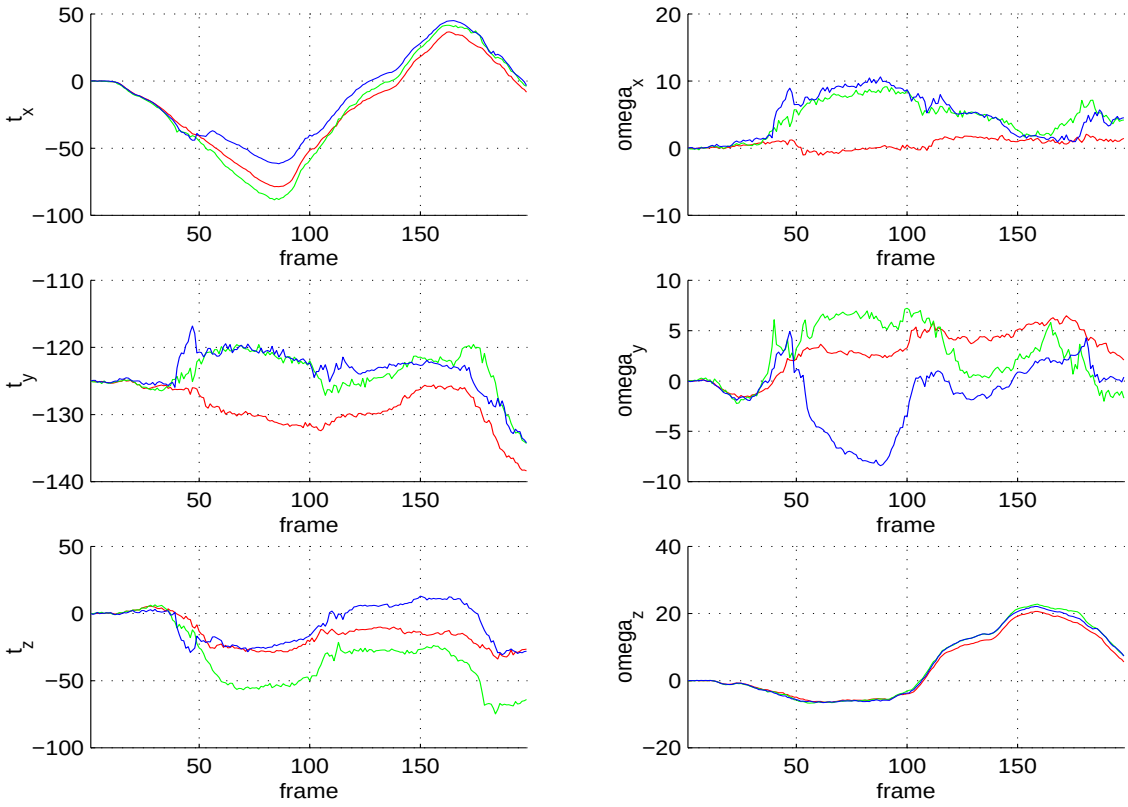

FIGURE 3.2.4. Comparison of the tracking results of the real sequence with different head models: red is the ellipsoidal model, green is the superellipsoidal model and finally blue is the detailed head model.

motion of the head. The model of the head that seems to be affected most by varying lighting conditions is the ellipsoidal one.

CHAPTER 4

Conclusions and Future Work

4.1. Conclusions

The algorithm presented in the previous chapters shares the architecture of most of the recent HTS; the optimization step is based on the matching of some feature points that are tracked along the video sequence. Those feature points are chosen according to an optimality criterion that allow to monitor the quality of the features and to track them reliably from one frame to the other. There is no need for some particular features to be visible over the entire video sequence: when a feature is lost the algorithm itself takes care of selecting a new one. The algorithm uses a fully perspective model for the camera and recovers all the six parameters that describe the pose of the head in the 3D space using just one single and uncalibrated camera. The computational cost of the routines that compose the algorithm is such that a real time implementation seem to be feasible. The results obtained by the experiments indicate that the algorithm is able to track quite accurately the trajectory of the head, also when large motions are involved and varying lighting conditions are present. A crucial point is the choice of a good 3D model for the head: the ellipsoidal model produces reasonable results if the accuracy is not the fundamental need, whereas the superellipsoidal model and the detailed model strongly improve the performance of the algorithm. Result seem to improved also using a reasonably large number of the features, even though this may affect the speed of the algorithm itself.

4.2. Future Work

4.2.1. The algorithm. The first thing we would like to do is to implement a real time version of the algorithm. This would allow to insert the head tracking algorithm in a more complex framework, like those addressed in (1). We also would like to carry out the initialization step in an automatic way. It would be also interesting to extend the algorithm in order to track not only the head but also more complex articulate objects, like the torso and for example the arm. The main problem that we expect to encounter in this generalization is the fact that the area (on the image plane) of the portion of the image corresponding to each of the body segments we wish to track is in general smaller than the area

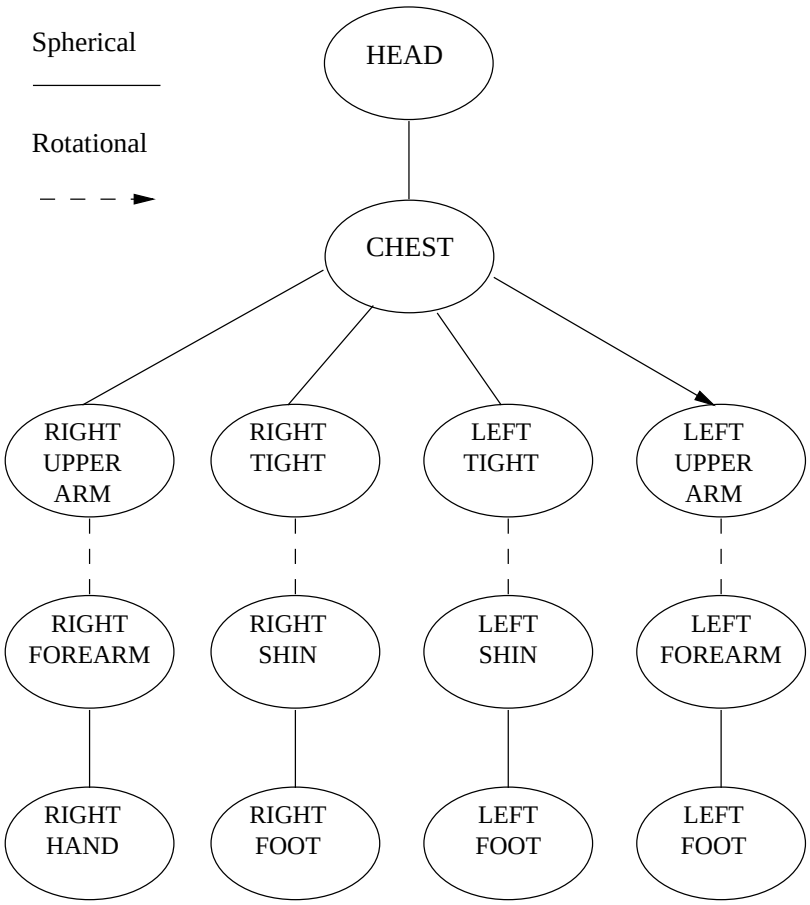

FIGURE 4.2.1. Kinematic chain used to describe the human body

corresponding to the head in a video sequence of the head only. This translates directly in a lower number of feature points that can be used for each body segment: this fact is likely to affect negatively the overall accuracy of the results. Nevertheless the possibility to describe such a situation through a kinematic chain (see [8] and figure (4.2.1)) composed by n elements, where only one element has 6 d.o.f. (for example the head), whereas the remaining $n-1$ elements have at most 2 d.o.f. (in fact all the segments of the human body are connected one to the other through spherical or rotational joints), reduces the number of variables involved in the optimization task and hopefully this will improve the accuracy of the results..

We also would like to explore which are the benefits that can be obtained using more than just one camera: in this case we need to retrieve the pose of each camera with respect to a common coordinate system (camera calibration), but we do not need to solve the correspondence problem for the different sets of features tracked on the different image planes.

Finally a few words concerning the head model. Detailed head model seem to improve greatly the performance of the algorithm, but in order to use them with different heads we must be able to parameterize the shape of those models so that they can be easily fitted with different heads. Moreover, to improve the results that may be obtained using a simple ellipsoidal or superellipsoidal model we would like to implement a confidence map w, where (using the conventions of (2.2.2)):

$$w\left(u,v\right):\left[0,2\pi\right]\times\left[0,\pi\right]\rightarrow\left[0,1\right]$$

measures the discrepancy of the model with the actual head through the weight w.

4.2.2. The experiments. The possibility to test the algorithm on a large number of real video sequences, where the ground truth values are known (reproducing the experimental setup described in [**11**]), will allow us to obtain precise quantitative information as far as the performance of the algorithm is concerned.

Bibliography

[1] R. Hartley and A. Zisserman, "Multiple View Geometry in Computer Vision", Cambridge University Press, 2000

[2] D. A. Forsyth and J. Ponce, "Computer Vision - A Modern Approach", Prentice Hall, 2001

[3] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, "Numerical Recipes in C: the Art of Scientific Computing", Cambridge University Press, 1998

[4] "Optimization Toolbox Users's Guide", The Mathworks Inc., 2000

[5] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with Application to Stereo Vision", IJCAI, 1981

[6] J. Shi and C. Tomasi, "Good features to track", Proc. IEEE CVPR, pp. 593-600, 1994

[7] J. Y. Bouguet, "Pyramidal Implementation of the Lucas Kanade Feature Tracker- Description of the Algorithm"

[8] C. Bregler and J. Malik, "Tracking People With Twists and Exponential Maps", Proc. IEEE CVPR, 1998

[9] R. Happee, M. Hoofman, A.J. van der Kronenberg, P. Morsink and J. Wismans, "A mathematical human body model for frontal and rearward seated automotive impact loading", STAPP paper, 98S-36, 1998

[10] S. Basu, I. Essa and A. Pentalnd, "Motion Regularization for Model-Based Head Tracking", Proc. ICPR, 1996

[11] M. La Cascia, S. Scarloff and V. Athitsos, "Fast, Reliable Head Tracking under Varying Illumination: an Approach Based on REgistration of Texture-Mapped 3D Models", Trans. IEEE PAMI, Vol. 22, No. 4, April 2000

[12] F. Preteux and M. Malciu, "Model-Based Head TRacking and 3D Pose Estimation", Proc. SPIE, Vol 3457, July 1998, pp. 94-110

[13] T. Otsuka and J. Ohya, "Real-time estimation of head motion using weak perspective epipolar geometry", Proc. of Fourth IEEE Workshop on Applications of Computer Vision, October 1998, pp. 442-453

[14] R. Cipolla and P. Gilbin, "Visual Motion of Curves and Surfaces", Cambridge University Press, 2000

[15] J. C. Lagarias, J. A. Reeds, M. H. Wright, P. E. Wright, "Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions", SIOPT Vol. 9, No.1, pp.112-147, 1998

APPENDIX A

The Optical Flow Based Head Tracking Algorithm

In chapter (2) we analyzed the projection-back projection algorithm which is based on the matching of a set of features that are tracked from one frame to the other. Since the performance of the method degrades when the number of the feature points is small, we must face the following problem: what happens if the number of the features we are able to track is too low[1]? The idea to solve this problem is to build a cost functional similar to (2.5.1), but that takes into consideration the optical flow instead just the position of the features. Since the optical flow can be computed without tracking explicitly one feature from one frame to the other, is reasonable to expect that the amount of information is some way "more dense".

A.1. From the Optical Flow to the Cost Functional

Let's consider the area of the image inside the apparent contour γ and let's divide that area in a set of disjoint windows W_k, $1 \leq k \leq N$, and let q_k be the central pixel of these windows. For each of those windows we can construct an equation like (2.3.8):

$$C_k \cdot \begin{pmatrix} u_k \\ v_k \end{pmatrix} + D_k = 0$$

At the same time, if we call $p(t)$ the set of all the points that represent the centroid of the window, then we can build also the set P, composed by the projection of the elements of $p(t)$ on the 3D ellipsoid. The algorithm now has a structure similar to the previous one: first of all we want to recover the estimation of the optical flow based on the back projection of the windows centroids. Since

$$q_{k,bp}\left(\phi\left(t+1\right)\right) = \mathcal{C}\left[P \cdot G_{w2c} \cdot \widehat{G}_{o2w}(t+1) \cdot Q_k\right]$$

we can define the vector:

$$\begin{pmatrix} u_k \\ v_k \end{pmatrix} = q_{k,bp}\left(\phi\left(t+1\right)\right) - q_k(t) \tag{A.1.1}$$

[1]This may happen for two reasons: the first one is that the area of the image that represents the head is small, and then the number of the windows W_k (see **??**) that can be build is low, and the second one is that the quality of the features is not good enough to allow them to be robustly tracked from one frame to the other: large areas of skin may be poor of texture.

and consequently the cost functional:

$$J\left(\left(\phi\left(t+1\right)\right)\right) = \sum_{k=1}^{N} w_k \left\| C_k \cdot \begin{pmatrix} u_k \\ v_k \end{pmatrix} + D_k \right\|^2 \tag{A.1.2}$$

In this case the structure of the weighting function w_k is immediately suggested by the Kanade, Lucas and Tomasi tracking algorithm: let Λ be the set of the minimum eigenvalues of C_k:

$$\Lambda = \{min\ \sigma(C_k) \mid 1 \leq k \leq N\}$$

and let $\lambda_{min} = min\ \Lambda$, $\lambda_{max} = max\ \Lambda$. Thus we define:

$$w_k = \frac{min\ \sigma(C_k) - \lambda_{min}}{\lambda_{max} - \lambda_{min}} \tag{A.1.3}$$

This means that windows that contain a pattern that can be easily tracked will be weighted more in the estimation of the parameter vector ϕ, carried out minimizing the cost functional (A.1.2). It is interesting to notice that many approaches that use the matching of the optical flow directly plug the expression of the displacement vector (A.1.1) in the BCCE (2.3.5) (see for example [**4**]). This might be considerably expensive as far as computational issues are concerned. The possibility to use windows represents a good trade-off between the needing for tracking accuracy and computational speed. We can also observe that, when the dimensions of W_k are unitary (so that W_k degenerates to a single pixel), the equation (2.3.8) is equivalent to (2.3.5)[2]. As far as the optimization algorithm is concerned please refer to (2.5).

A.2. The Results

We did not carry out systematic experiments using this method, but some of the test we have done suggest that its performance is generally worse than the performance obtained using the feature matching algorithm. In particular this method seem to have many problems when the velocity of the points on the head is orthogonal to the image plane. This conclusions are supported also by the paper by Preteux and Malciu (see [**12**] and (1.3.3)) where the authors are compelled to combine the cost functional derived from the optical flow matching with a cost functional related to the texture matching in order to obtain accurate results.

[2]Actually, in this case, we can't use the expression (A.1.3) for the weighting function, since $\lambda_{min} = \lambda_{max} = 0$ for each window. This is a consequence of the fact that a single pixel doesn't carry any information about texture.