CHAPTER II

TRACKING AND THE 2-DAFT SOFTWARE FRAMEWORK

In this chapter, we define some terms related to articulated figure tracking and place our tracking approach within the existing body of tracking research. Then we go on to describe 2-DAFT – short for "2-D Articulated Figure Tracker" – a software framework we have developed for tracking.

2.1 Articulated Figure Tracking

An articulated figure is a figure made up of a number of connected "links," each of which can move relative to the other links. The human body is an articulated figure where the links are the torso, head, and limbs. Human figure tracking is therefore a special case of articulated figure tracking. Articulated figure tracking stands in contrast to "rigid body" tracking, where all parts of the tracking target remain rigid with respect to one another.

There are a great variety of approaches to articulated figure tracking. In this section we will define some terms related to tracking, review some of the approaches to figure tracking, and place our own approach in the context of the field.

Some tracking techniques use multiple videos from different views to try to recover the body pose in 3-D [2] or just to improve recovery of the 2-D pose [16]. Others use just a single view of the motion. We are concerned with the latter technique: tracking from monocular video.

A kinematic model describes the degrees of freedom of the links and the connections between them. For example, a kinematic model of the human body might dictate that each joint can rotate in three dimensions, and that the foot is connected to the tibia which is connected to the femur and so on. A kinematic model serves two purposes. First, it describes the desired description of the body pose. If the kinematic model describes the degrees of freedom as one 3-D rotation per joint, then we are saying that we want to recover the body pose as a list of joint angles. Second, the kinematic model imposes constraints on the motion of the figure. If the tibia is connected to the femur, then wherever the tibia goes the femur must follow. Other constraints, such as “the knee can only bend backwards” may also be included in the kinematic model, although often they are not.

A variety of kinematic models for both 2-D [6] and 3-D [9] tracking have been used. We use the Scaled Prismatic Model (SPM) [13], which is a “2.5-D” kinematic model. We say “2.5-D” because although an SPM is embedded in 2-D, it models the foreshortening caused by limbs rotating towards or away from the camera by allowing the length of links to shrink or grow. (Note that this makes the name 2-DAFT a misnomer; technically, it should be 2.5-DAFT. But a little inaccuracy is a small price to pay for a catchy acronym.) Figure 2.4 illustrates two SPM links. The SPM model will be described more in Section 2.2.1.

An appearance model describes the appearance of the tracking target. A very simple appearance model is a patch-based model, where the appearance of each link is described as a patch of pixels. These patches can be copied from the regions of one frame of the video corresponding to each link. Another alternative is to use a contour-based model. In this case, the appearance of each link is described by its silhouette. This silhouette can be described parametrically. Alternatively, an edge detector may be applied to one frame of the video to generate an edge image. The edge pixels associated with each link can then be copied from the edge image.

2-DAFT allows for the implementation of different types of appearance models. In the experiments presented here, we have used a contour-based appearance model. Section 2.2.2 describes our appearance model in greater detail.

Most tracking techniques take a “top-down” approach. That is, they estimate the overall configuration of the kinematic model and then use that estimate as the starting point for a local optimization in the state space of the kinematic model that attempts to match the appearance model to features in the image. This is the approach 2-DAFT takes. However, some recent work, e.g. [12, 15], takes a “bottom-up” approach. First, they apply head, torso, and limb detectors to detect likely locations of body parts in the image. They then combine the information from these detectors to make a hypothesis about the configuration of the entire body. This approach has certain advantages – such as the ability to initialize itself – and is gaining popularity, but we will not deal with it here.

Besides the top-down versus bottom-up question, another question any tracking algorithm must answer is how to represent the probability density function of the body pose given the image data, and how to propagate such probabilistic information from frame to frame. A classical approach to this problem is the Kalman filter. However, the Kalman filter is limited to dealing with unimodal distributions. The properties of real-world scenes – e.g. background clutter and occlusions – typically lead to a multimodal distribution on the probability of the body pose. This can be explained in simple terms by observing that several different regions of the image may resemble the appearance model. One approach to representing multimodal densities is multiple hypothesis tracking, which was applied to figure tracking in [1]. Another common technique is the use of Monte Carlo methods, such as Isard and Blake's CONDENSATION algorithm [5].

Since we are mainly interested in investigating properties of the dynamic model, we adopt a very simple, unimodal representation of the probability density.

Having given an overview of articulated figure tracking in general and our approach to human figure tracking in particular, we now go into detail about the 2-DAFT framework.

2.2 The 2-DAFT Software Framework

2-DAFT (2-D Articulated Figure Tracker) is a software framework for implementing monocular, 2.5-D articulated figure tracking algorithms. 2-DAFT takes an object-oriented approach, dividing functionality and data involved in tracking into a number of different modules. This makes 2-DAFT flexible, allowing the implementation of tracking algorithms using various kinematic models, image features, and dynamic models with a minimum of new coding.

2-DAFT consists of five main modules: kinematic model, appearance model, image registration, dynamic model, and control/GUI module. Figure 2.1 shows the relationships between the different modules. An arrow from module A to module B indicates that A

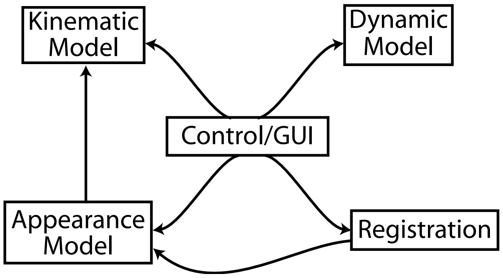


Figure 2.1: 2-DAFT modules

makes function calls to B. In the following sections, we describe each of these modules in turn.

2.2.1 Kinematic Model

2-DAFT's kinematic models consist of a number of Scaled Prismatic Model (SPM) links. A "kinematic tree" describes the connections between the links in the articulated figure. We refer to the link at the top of the tree as the "root" link. Figure 2.2 shows an example of a kinematic tree for tracking a human body. In this case, the torso is the root link.

The entire kinematic tree is embedded in a "world" coordinate frame. In addition, each link has its own local coordinate frame. The x-axis of the local coordinate frame lies along the length of the link, and the y-axis lies along the width of the link (see Figure 2.3). Points in the link coordinate frame are specified in relative coordinates. (0.0,0.0) is defined to be the center of the base of the link. 1.0 is defined to be equal to the length of the link on the x-axis, and one-half the thickness of the link on the y-axis. See Figure 2.3 for an illustration.

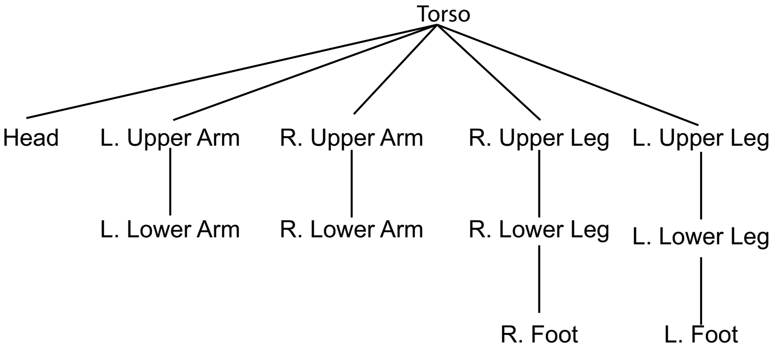


Figure 2.2: A kinematic tree structure for the human body

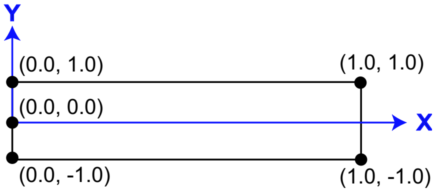


Figure 2.3: The local, relative coordinate system of a link

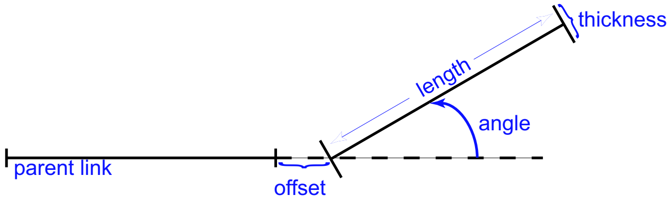


Figure 2.4: The degrees of freedom of a link

Each link can have up to five degrees of freedom: rotation, length, thickness, and x,y offset. Rotation is specified as an angle relative to the parent link (except for the root link, for which rotation is specified relative to the world coordinate frame). Length, specified in pixels, is a scaling factor that scales the link along its x-axis; similarly, thickness scales the link along its y-axis. Offset gives the location of the link relative to its parent. Offset consists of an x and y translation. For the root link, these are specified in the world coordinate frame. For other links, the offset is specified in relative coordinates in the parent's coordinate frame. See Figure 2.4 for an illustration of a link's degrees of freedom.

2.2.1.1 File Format

2-DAFT uses a simple text file to specify kinematic models. By way of example, Figure 2.5 shows a file that specifies a kinematic chain with two links: a root link and a child link.

In a kinematic tree file, the name of a link is followed by the link's specification enclosed in braces. The specification includes a number of parameters, as well as the specifications of any children of the link. The parameters are organized into three blocks.

The first block consists of boolean (1 or 0) parameters that specify which of the possible

```
Root {
    1   % isStateAngle
    1   % isStateLen
    0   % isStateThickness
    1   % isStateOffset

    1   % canSetAngle
    1   % canSetLen
    1   % canSetThickness
    1   % canSetOffset

    0   % default angle
    50  % default len
    20  % default thickness
    300 % default offset x
    300 % default offset y

    Link2 {
        1   % isStateAngle
        1   % isStateLen
        0   % isStateThickness
        0   % isStateOffset

        1   % canSetAngle
        1   % canSetLen
        1   % canSetThickness
        1   % canSetOffset

        0.0  % default angle
        30   % default len
        10   % default thickness
        1.0  % default offset x
        0.0  % default offset y
    }
}
```

Figure 2.5: Sample kinematic tree file

degrees of freedom are part of the state of this link.

The second block also contains boolean parameters that relate to the degrees of freedom of the link. This time, though, the parameters specify whether or not the corresponding degree of freedom can be "set." A degree of freedom that "can be set" is one which the user may modify when he or she is setting the initial state or setting the appearance model (see Section 2.2.2). However, only those degrees of freedom which are part of the state will be modified during image registration (see Section 2.2.3).

The final block specifies initial values for all of the degrees of freedom (regardless of whether or not they are part of the state). The default angle is given in radians and is relative to the parent link's angle (or to the world coordinate frame for the root link). The default length and thickness are given in pixels. For the root link, the default x and y offsets are specified in pixels in the world coordinate frame. For other links, the default x and y offsets are specified in relative coordinates in the frame of their parent link. For instance, in Figure 2.5, Link2's offset of (1.0, 0.0) will place its origin at the endpoint of its parent, Root.

2.2.1.2 Kinematic Jacobian

In order to conduct image registration (see Section 2.2.3), it will be necessary to compute the kinematic Jacobian; that is, the matrix:

$$\begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \cdots & \frac{\partial x}{\partial \theta_n} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \cdots & \frac{\partial y}{\partial \theta_n} \end{bmatrix}$$

where x and y are the world coordinates of a point attached to one of the links and the θ_i are the degrees of freedom of all the links. The kinematic Jacobian describes the change in the location of the point (x, y) due to a change in the state of the kinematic model. Note that the global position of a point (x, y) on link i will be influenced by changes in the state of all links in the chain between i and the root link, including i and the root link. The position of (x, y) will *not* be affected by changes in the state of any other links. Hence the columns in the kinematic Jacobian that correspond to the state of links not in the chain between i and the root will contain zeroes.

In the remainder of this section we derive expressions for the partial derivatives of x and y with respect to the various degrees of freedom. These formulas are partially based on [13].

First, we note the equations for calculating x and y, the world coordinates of the point we are interested in obtaining a kinematic Jacobian for:

$$x = lx_l cos(\theta_w) - \frac{t}{2} y_l sin(\theta_w) + o_x \tag{2.1}$$

$$y = lx_l sin(\theta_w) + \frac{t}{2} y_l cos(\theta_w) + o_y \tag{2.2}$$

where:

l = length of link i, in which the point is embedded

x_l, y_l = local, relative x- and y-coordinates of the point in link i

θ_w = rotation, relative to the world frame, of link i

t = thickness of link i

o_x, o_y = x- and y-coordinates of the origin of link i in world coordinates

By differentiating (2.1) and (2.2), we can find expressions for the various partial derivatives.

For rotation degrees of freedom, where θ_i represents the rotation of link i, we have:

$$\begin{array}{rcccc} \frac{\partial x}{\partial \theta_i} & = & -lx_l sin(\theta_w) - \frac{t}{2} y_l cos(\theta_w) & = & -y + o_y \\ \frac{\partial y}{\partial \theta_i} & = & lx_l cos(\theta_w) - \frac{t}{2} y_l sin(\theta_w) & = & x - o_x \end{array}$$

The formulas for the effect on (x, y) due to links up the chain between i and the root are the same.

For length degrees of freedom, we have:

$$\frac{\partial x}{\partial l_i} = x_l cos(\theta_w) \qquad \frac{\partial y}{\partial l_i} = x_l sin(\theta_w)$$

For link j, where j is a link between i and the root and $j + 1$ is the immediate successor of j on the chain between j and i, the formulas are the same except that x_l is replaced by the x-coordinate of the position of $j + 1$'s origin in j's coordinate frame.

For thickness degrees of freedom, we have:

$$\frac{\partial x}{\partial t_i} = -\frac{1}{2} y_l sin(\theta_w) \qquad \frac{\partial y}{\partial t_i} = \frac{1}{2} y_l cos(\theta_w)$$

A similar situation as for lengths applies for links j: the equations are the same, but y_l is replaced by the y-coordinate of the position of $j+1$'s origin in j's coordinate frame.

For offset degrees of freedom, we have a special case for the root link, since its offset is given in global coordinates while other links' offsets are given in local coordinates of their parent links. For offsets (o_x, o_y) in the root, we have:

$$\begin{array}{ll} \frac{\partial x}{\partial o_x} = 1 & \frac{\partial y}{\partial o_x} = 0 \\ \frac{\partial x}{\partial o_y} = 0 & \frac{\partial y}{\partial o_y} = 1 \end{array}$$

For non-root links i, the appropriate partial derivatives depend on the length (l_k) and global rotation $(\theta_{w,k})$ of i's parent link, k.

$$\begin{array}{lllllll} \frac{\partial x}{\partial o_{x,i}} & = & l_k cos(\theta_{w,k}) & & \frac{\partial y}{\partial o_{x,i}} & = & l_k sin(\theta_{w,k}) \\ \frac{\partial x}{\partial o_{y,i}} & = & -\frac{1}{2} t_k sin(\theta_{w,k}) & & \frac{\partial y}{\partial o_{y,i}} & = & \frac{1}{2} t_k cos(\theta_{w,k}) \end{array}$$

The same equations hold for links j on the path between i and the root.

2.2.1.3 Code Structure

Two classes in 2-DAFT implement the kinematic model. `SPMLink` stores the parameters of each link as well as pointers to its parent and children. `KinematicTree` stores all of the links in an array. Most calls into the kinematic model code are made to `KinematicTree`.

2.2.2 Appearance Model

`AppearanceModel` is an abstract 2-DAFT class that defines the interface for appearance models. New appearance models can be created by writing subclasses of `AppearanceModel`. In our experiments, we used a contour-based appearance model, implemented in the class `EdgeAppearanceModel`.

The appearance models we have implemented so far use static features, although it would be possible to implement a dynamic appearance model within the 2-DAFT framework.

2.2.2.1 Appearance Model Initialization

Most static appearance models require a manual initialization step. To demonstrate 2-DAFT's initialization algorithm, we will give an example of initializing an edge appearance model.

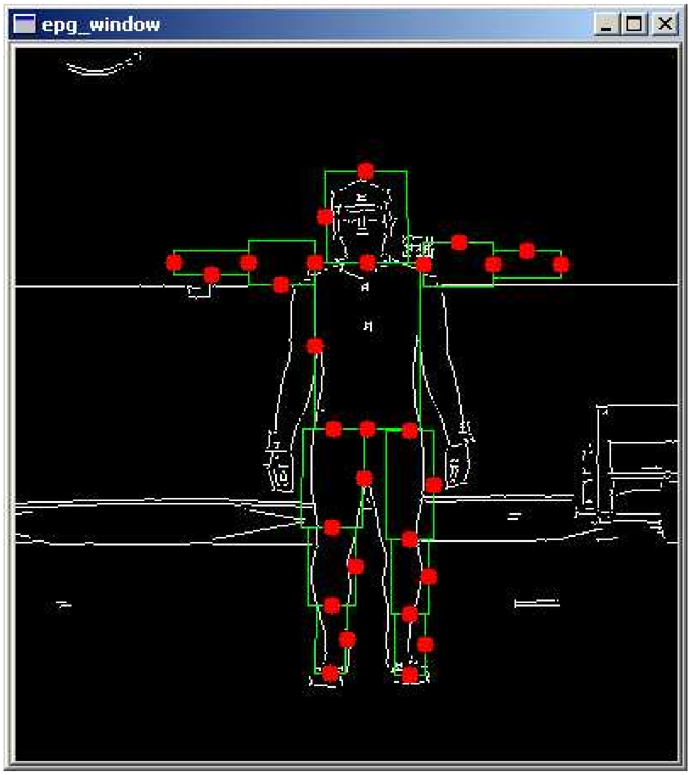


Figure 2.6: Initializing the appearance model

First, the user specifies a frame of the input video to be used for model initialization. 2-DAFT runs an edge detector on the specified frame, then displays the edge image in a window. On top of the edge image, an outline of each link in the kinematic model is superimposed (Figure 2.6). Initially, the configuration of the kinematic model is defined by the initial values given in the kinematic tree file (see Section 2.2.1.1). The user can use the mouse to modify the kinematic state by dragging the circular control points so that the links are each aligned with the corresponding part of the image.

Next, 2-DAFT copies the region of the image covered by each link. It displays a window containing the edge pixels in each link one by one. At this point, the user can use the mouse to turn individual pixels on and off. This can be used to eliminate background edges that should not be considered part of the appearance model.

Finally, for each link 2-DAFT stores the locations of that link's edge points in the link's coordinate system. Now, given a kinematic model state, 2-DAFT can map the appearance

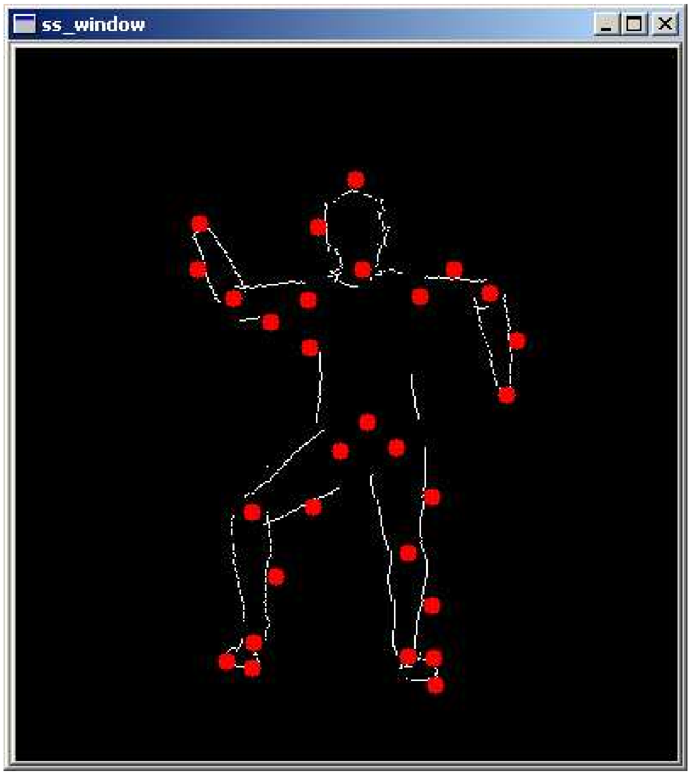


Figure 2.7: Transforming the appearance model to a new configuration

model into the image, as in Figure 2.7.

2.2.2.2 The Residual and Its Jacobian

2-DAFT's standard registration algorithm seeks to minimize the objective function $R^T R$, where R is the residual vector $[r_1, r_2, \ldots, r_m]^T$. The definition of the residual vector depends on the appearance model. For the edge appearance model, there is one residual per point in the model. Given a target image, each residual is defined to be the "chamfer distance" of a model point transformed according to the kinematic state. The chamfer distance is the distance to the nearest detected edge pixel in the target image.

The minimization algorithm requires the computation of the Jacobian of the residual:

$$J = \begin{bmatrix} \frac{\partial r_1}{\partial \theta_1} & \frac{\partial r_1}{\partial \theta_2} & \cdots & \frac{\partial r_1}{\partial \theta_n} \\ \frac{\partial r_2}{\partial \theta_1} & \frac{\partial r_2}{\partial \theta_2} & \cdots & \frac{\partial r_2}{\partial \theta_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial r_m}{\partial \theta_1} & \frac{\partial r_m}{\partial \theta_2} & \cdots & \frac{\partial r_m}{\partial \theta_n} \end{bmatrix}$$

Let $p_i(\theta)$ be the x,y-position, in global coordinates, of point i in the appearance model. p_i is a function of θ, the state of the kinematic model. r_i, the ith residual, is a function of position: $r_i(p_i(\theta))$. We can use the chain rule to break the partial derivatives in J into two pieces:

$$\nabla_\theta(r_i(p_i(\theta))) = (\nabla_{p_i} r_i)^T \cdot \nabla_\theta p_i = \begin{bmatrix} \frac{\partial r_i}{\partial x_i} & \frac{\partial r_i}{\partial y_i} \end{bmatrix} \begin{bmatrix} \frac{\partial x_i}{\partial \theta_1} & \frac{\partial x_i}{\partial \theta_2} & \cdots & \frac{\partial x_i}{\partial \theta_n} \\ \frac{\partial y_i}{\partial \theta_1} & \frac{\partial y_i}{\partial \theta_2} & \cdots & \frac{\partial y_i}{\partial \theta_n} \end{bmatrix}$$

This decomposition allows us to neatly divide the computation of J between two modules. The first part, $\nabla_{p_i} r_i$ is computed by the appearance model, using finite differences. The second part, $\nabla_\theta p_i$, is the kinematic Jacobian described in Section 2.2.1.2 and is computed by the kinematic model.

In order to compute each row of the residual Jacobian given an image, the appearance model computes $\nabla_\theta p_i$ for each i. It then calls the kinematic model to get $\nabla_\theta p_i$. The Jacobian row is the product of these two matrices.

2.2.3 Registration Module

Given an image, a kinematic model with associated state, and an appearance model, the registration module performs a local search in the kinematic model state space to try to match the appearance model to the image as closely as possible. It does this by minimizing the score, $R^T R$, of the appearance model given the kinematic state and image. The registration module conducts this minimization using the Levenberg-Marquardt method ([11], pp.542f). Given a state x, the Levenberg-Marquardt method calculates a descent direction s as

$$s = (A + \epsilon I)^{-1} \nabla$$

where A is the Hessian of the objective function evaluated at x and ∇ is the gradient of the objective function evaluated at x. In this case the objective function is $R^T R$. Thus the Hessian is $J^T J$, where J is the residual Jacobian described in Section 2.2.2, and the gradient is $J^T R$. Making these substitutions we have:

$$s = (J^T J + \epsilon I)^{-1} J^T R$$

In the normal Levenberg-Marquardt method we would choose $x_+ = x + s$. However, in this case we find it works better to use s as a direction, and conduct a golden ratio search ([11], pp.293f) in the direction s. Golden ratio search is a line search algorithm that finds the minimum value along the line from x to $x + s$. The next x value in the iteration is chosen to be the result of the golden ratio search.

This procedure – the calculation of s followed by a line search – is repeated until the value of the gradient at x is reduced sufficiently or a limit on the number of iterations is reached.

2.2.4 Dynamic Model

Given an observation about the current state of the kinematic model, and perhaps some other information gleaned from past observations, the dynamic model predicts the state at the next time instant. The 2-DAFT class `Dynamics` is an abstract class that defines the interface for a 2-DAFT dynamic model. To define a new dynamic model that works with existing code, one simply has to create a new subclass of `Dynamics`. A number of such subclasses have already been written.

The interface of `Dynamics` is designed to parallel the predictor-corrector interpretation of the Kalman filter, although there is no need to actually use a Kalman filter – this structure can accommodate different types dynamic models. `Dynamics` contains four main methods:

`estimateX0(ys)`

Initializes or estimates the initial hidden state variable, possibly using the parameter `ys` which will normally be a list of observations from different frames.

`predict()`

Predicts the value of the hidden state variable at the next time step.

`correct(y)`

Based on the parameter `y`, an observation of the state, correct the prediction `x` of the hidden state.

`observe()`

Return an estimate of what the current observation should be, based on the current estimate of the hidden variable.

As an example, let's look at how these methods would be defined for a Linear Dynamic System with Kalman filtering and for a constant velocity model.

First, for a Linear Dynamic System:

`estimateX0(ys)`

> `ys` is a list of the observations from the first few frames. `x`, the hidden state of the LDS, can be estimated using `ys` and the impulse response of the system.

`predict()`

> The next hidden state is predicted by `x = A*x`, where `A` is the plant matrix of the LDS. The error covariance matrix prediction is also calculated here.

`correct(y)`

> Given the observation `y`, the Kalman gain matrix is calculated and the hidden state `x` is corrected according to the standard Kalman filter equations. The error covariance matrix estimate is also corrected.

`observe()`

> Return an estimate of observation calculated as `C*x`, where `C` is the observation matrix of the LDS.

Now we consider how a constant velocity model might fit into this framework. In a constant velocity model, the velocity from frame i to frame $i+1$ is assumed to be the same as that observed from frame $i-1$ to frame i.

`estimateX0(ys)`

> `ys` is a list of the observations from the first two frames. In this case, the "hidden state" is the estimated velocity, which is calculated as `vel = ys[1] - ys[0]`. The observation from the most recent frame, `ys[1]`, is also saved as `yc`.

`predict()`

> The next observation is predicted by adding the expected velocity to the previous observation: `yn = yc + vel`.

`correct(y)`

> Given the observation `y` from the current frame, calculate the new velocity, `vel =`

`y - yc`. Set `yc = y`.

`observe()`

Return the last observation, `yc`, if `predict()` has not been called yet, or return the estimate of the next observation, `yn`, if `predict()` has been called.

How does the SLDS model fit into this framework? We could imagine creating an `SLDSDynamics` class that would include several `Dynamics` objects, one for each of the LDS's in the SLDS. It would also include other data and functionality needed by an SLDS.

2.2.5 Control Module

The control module contains the code that knits the other modules together to form a tracking algorithm. We also include much of the GUI code under the heading of "control module." 2-DAFT's control code is mostly encapsulated in the `Project` (for "tracking project") class. The `Project` class also includes code for setting various tracking parameters and viewing the output of the tracking process.

Each tracking application built using 2-DAFT will have its own implementation of the `Project` class. The `Project` class will contain the meat of the tracking algorithm. This includes, for example, building and propagating a representation of the probability density function on the state space, and estimating the kinematic model configuration in each frame. The 2-DAFT source code includes a "generic" `Project` class that can be customized, usually with minimal effort. We have implemented a number of tracking applications using 2-DAFT, each with its own `Project` class.

Figure 2.8 describes a simple, generic tracking algorithm and shows how the various modules work together in tracking. This algorithm, or something like it, would be implemented in the `Project` class for a particular tracking application.

2.2.5.1 GUI

In this section we highlight some of the aspects of the generic GUI for a 2-DAFT application.

Figure 2.9 shows the 2-DAFT main dialog, which is displayed when a 2-DAFT application starts. The center column contains buttons for loading the kinematic model, loading

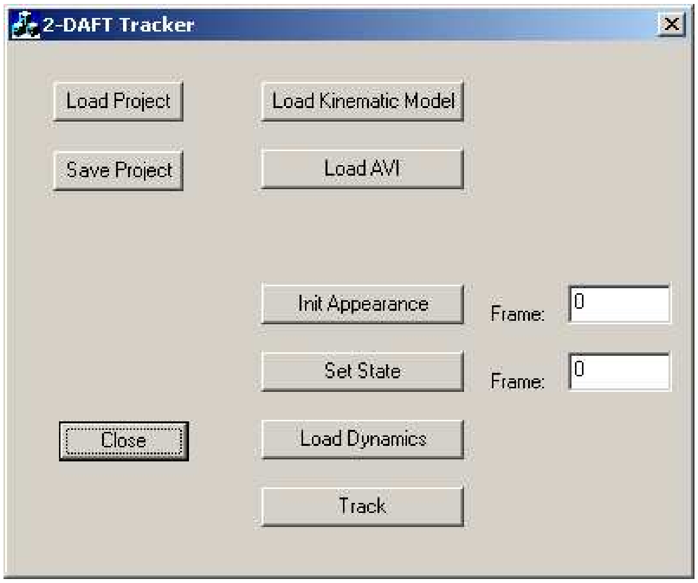
```
Allow the user to load a video, a kinematic model file, and the dynamic model parameters
Allow the user to initialize the appearance model
Allow the user to initialize the kinematic model configuration
n = # of frames in the video
For i = 2...n
    Get the current state, $\theta$, from the kinematic model.
    Call the dynamics model to predict $\theta$ for frame i.
    Load frame i.
    Call the registration module to match $\theta$ to the image.
    Treat the registration result as an observation; use it to correct the dynamic model's hidden variables.
    Call the dynamic model to get the current best estimate of $\theta$, $\theta'$. Set $\theta = \theta'$ in the kinematic model.
end
```

Figure 2.8: Generic tracking algorithm

the AVI video file to be tracked, initializing the appearance model (as described in Section 2.2.2), setting the initial state, and loading parameters of the dynamics model (e.g. the various matrices that define an LDS). On the left are buttons for loading and saving "projects." A project file contains all information about the setup of a tracking experiment: the kinematic model, video file name, appearance model description, etc. Project files make it easy to repeat an experiment without having to go through all the setup steps again.

The "Track" button (bottom center) brings up the tracking dialog shown in Figure 2.10. The four buttons grouped at the top of the dialog control the progress of the tracking algorithm. The "Step" buttons display results of registration one iteration at a time. The "Frame" buttons cause all iterations of image registration to be done at once. "Track All Frames" attempts to perform registration for every frame in the video, and writes the output to a video file specified by the user. Tracking results are not displayed in the GUI when this feature is used. "Save History" writes the estimated kinematic model configuration vector of each frame to a text file.

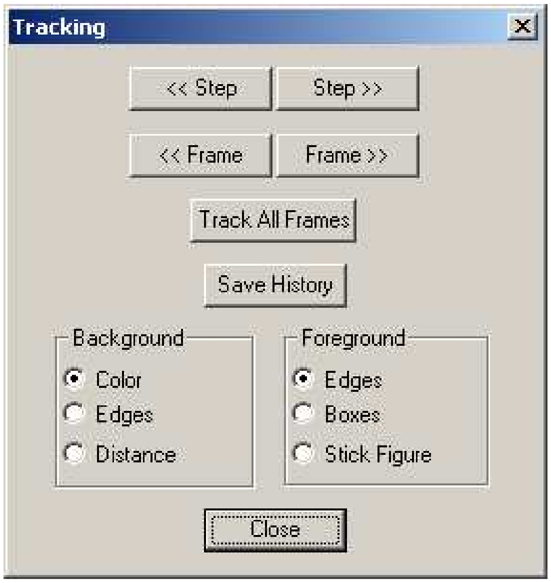
The "Background" and "Foreground" sections of the tracking dialog allow the user to specify the type of graphical output that is displayed as tracking progresses. Figure 2.11 shows some examples of the output options. To specify the output, the user chooses a



Frame: 0

Frame: 0

Figure 2.9: 2-DAFT main dialog



Background

● Color
○ Edges
○ Distance

Foreground

● Edges
○ Boxes
○ Stick Figure

Figure 2.10: 2-DAFT tracking dialog

background, which represents the current video frame, and a foreground, which represents the current estimated state of the kinematic model. The background options are:

`Color:` Display the original video frame.

`Edges:` Display the output of an edge detector applied to the video frame.

`Distance:` Display the chamfer distance image of the original video frame. The chamfer distance of a point in an edge image is the distance from that point to the nearest edge pixel. In the chamfer distance image, the lighter the color of a pixel the greater its chamfer distance.

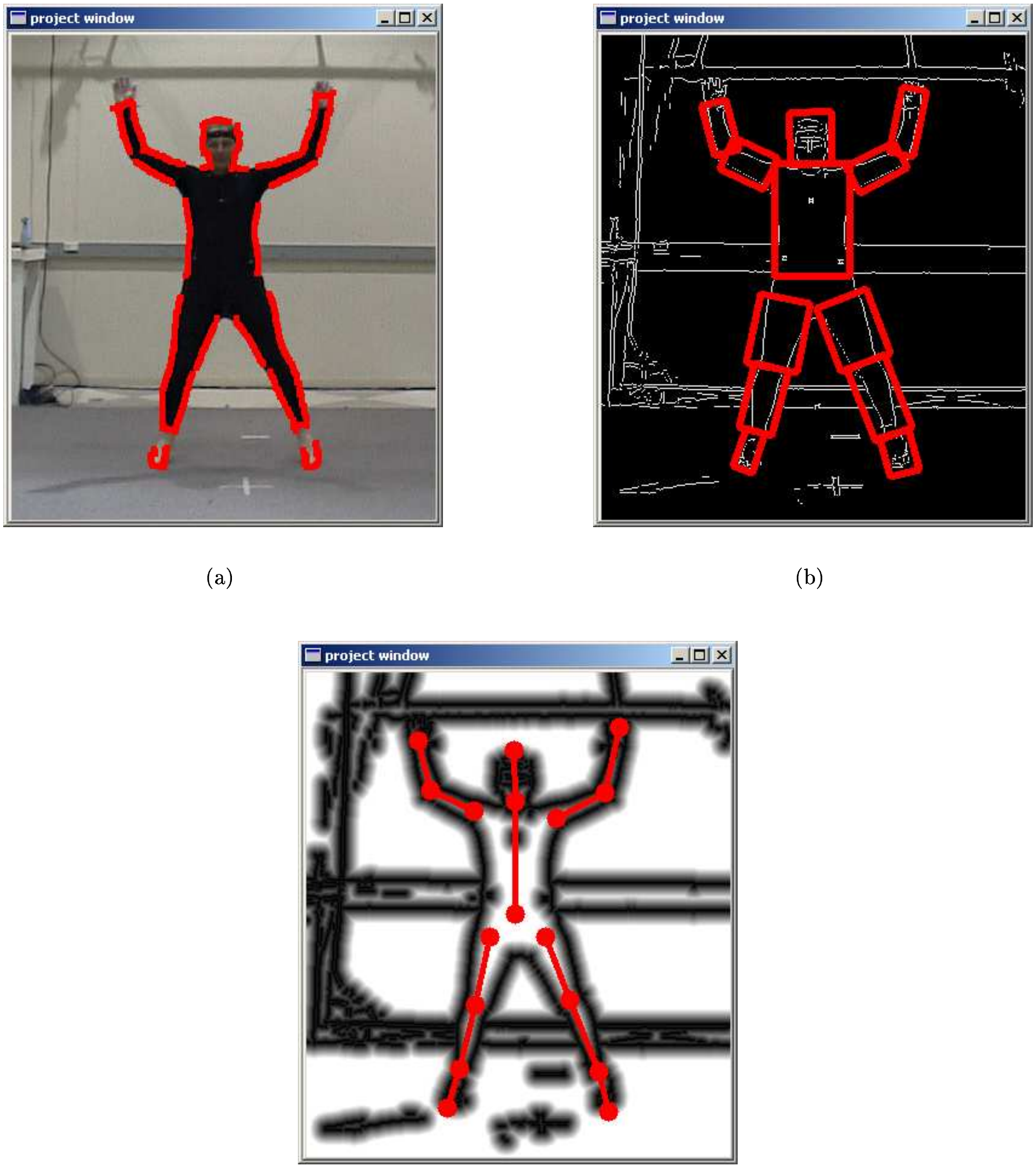
The foreground options are:

`Edges:` Display the edges of the appearance model, projected into the image according to the kinematic model state. (This option only makes sense when the edge appearance model is used.)

`Boxes:` Draw rectangles around the projected locations of each kinematic link.

`Stick Figure:` Draws a “stick figure” representation of the kinematic model: a circle is drawn on each joint center position, and lines connect joint centers.

These are the main features of the generic 2-DAFT GUI. Other features may be added as needed. Or, if desired, 2-DAFT may of course be used to build a “command-line” type program.



(a)

(b)

(c)

Figure 2.11: Some output options in 2-DAFT. (a) The original video frame, with the appearance model – in this cases the target edges – overlaid. (b) Output of an edge detector on the original frame, with boxes representing the location of kinematic links. (c) Chamfer distance map of original image, with a stick figure representation of the kinematic model configuration.

REFERENCES

[1] Cham, T.-J. and Rehg, J. M., "A multiple hypothesis approach to figure tracking," in *CVPR*, 1999.

[2] Gavrila, D. M. and Davis, L. S., "3-d model-based tracking of humans in action: a multi-view approach," in *CVPR*, 1996.

[3] Ghahramani, Z. and Hinton, G. E., "Switching state-space models," Tech. Rep. CRG-TR-96-3, Dept. Comp. Sci., University of Toledo, 6 King's College Road, Toronto M5S 3H5, Canada, 1998.

[4] Hodgins, J. K., Wooten, W. L., Brogan, D. C., and O'Brien, J., "Animating human athletics," in *SIGGRAPH*, 1995.

[5] Isard, M. and Blake, A., "Condensation – conditional density propagation for visual tracking," *The International Journal of Computer Vision*, vol. 29, no. 1, 1998.

[6] Ju, S., Black, M., and Yacoob, Y., "Cardboard people: a parameterized model of articulated motion," in *Int. Conf. on Automatic Face and Gesture Recognition*, 1996.

[7] Li, Y., Wang, T., and Shum, H.-Y., "Motion texture: a two-level statistical model for character motion synthesis," in *SIGGRAPH*, 2002.

[8] Overschee, P. V. and DeMoor, B., *Subspace Identification for Linear Systems: Theory-Implementation-Applications.* Kluwer Academic Publishers, 1996.

[9] Pavlović, V., Rehg, J. M., Cham, T.-J., and Murphy, K. P., "A dynamic bayesian network approach to figure tracking using learned dynamic models," in *ICCV*, 1999.

[10] Pavlović, V., Rehg, J. M., and MacCormick, J., "Learning switching linear models of human motion," in *NIPS*, 2000.

[11] Press, W. H., Vetterling, W. T., Flannery, B. P., and Teukolsky, S. A., *Numerica Recipes in C.* Cambridge University Press, 1990.

[12] Ramanan, D. and Forsyth, D., "Finding and tracking people from the bottom up," in *CVPR*, 2003.

[13] Rehg, J. M., Morris, D. D., and Kanade, T., "Ambiguities in visual tracking of articulated objects using two- and three-dimensional models," *The International Journal of Robotics Research*, vol. 22, June 2003.

[14] Shumway, R. H. and Stoffer, D. S., "Dynamic linear models with switching," *J. of the American Statistical Association*, vol. 86, Sept 1991.

[15] Sigal, L., Bhatia, S., Roth, S., Black, M., and Isard, M., "Tracking loose-limbed people," in *CVPR*, 2004.

[16] Sminchisescu, C. and Triggs, B., "Covariance scaled sampling for monocular 3d body tracking," in *CVPR*, 2001.

[17] Soatto, S., Doretto, G., and Wu, Y. N., "Dynamic textures," in *ICCV*, 2001.

[18] Wren, C. R. and Pentland, A. P., "Dynamic models of human motion," in *Proc. 3rd International Conference on Automatic Face and Gesture Recognition*, 1998.