Planning human walk in virtual environments

J. Pettré, T. Siméon, J.P. Laumond

LAAS/CNRS,
7, avenue du Colonel Roche, 31077 Toulouse Cedex 04 - France
{jpettre,nic,jpl}@laas.fr

Abstract

This paper presents a method for animating human characters, especially dedicated to walk planning problems. The method is integrated in a randomized motion planning scheme, including a steering method dedicated to human walk. This steering method integrates a character motion controller assuming realistic animations. The navigation of the character through a virtual environment is modeled as a composition of Bézier curves. The controller is based on motion capture data editing techniques. This approach satisfies some essential computer graphics criteria: a realistic result, a low response time, a collision-free motion in possibly constrained 3D environments. The approach has been implemented and successfully demonstrated on several examples.

1 Introduction

Animating human characters has become a very active research field since cinematographic and entertainment industries have demonstrated the greatness of the application field. Many animation techniques recently developed allow to reduce production time and to improve the realism [1]. Animating automatically virtual actors raises problems due to the complexity of the kinematic structure of the models and to the complexity of the real human behaviors. On one hand, realistic human motion controllers exist in computer animation literature[1, 2]. On the other hand, robotics has developed efficient algorithms over complex and constrained motion planning problems[3, 4]. Our contribution is to take advantage from both sides. We propose a solution for planning human walk through virtual environments, taking advantage of randomized motion planning algorithms, combined to a character motion controller based on motion capture editing techniques.

Walking virtual actors Through the character animation problem [1], walking is crucial for navigating in a virtual world. Synthesizing walk animations is traditionally approached by three techniques: kinematics (forward and inverse), dynamics and motion data based.
The kinematic approach is commonly dedicated to hand made animations, eventually assisted by specific interpolation techniques in order to generate transition positions. Inverse kinematics reduce the time for positioning the body, as the animator acts on the end effectors only. It also can be used as an animation filter in order to solve kinematic constraints violation [5]. A dynamic approach is described in [6]. Respecting the physical laws intrinsically increases the realism of the animation but a high computation power is needed. Motion capture data based techniques are presented in [7, 8, 9]. The data are fixed recorded motion sequences warranting realism[10]. Moreover, re-targeting is possible to animate different trajectories thanks to motion blending and warping. Motion blending [11] consists in interpolating parameters of several motions in order to produce new motions, whereas motion warping [7] consists in modifying a single motion in order to fit to a new trajectory. See [9, 2, 12] for a more detailed state of the art.

Path planning Motion planning algorithms are well-developed in the robotics literature [3]. Robot motion planning is concerned by: synthesizing a collision free motion in constrained environments, controlling systems with respect to the kinematic constraints [4] and by the achievement of complex tasks. Randomized motion planning is an efficient solution taking into account these constraints: a description is given in [13, 14].

Human motion planners Closing together motion planning algorithms issued from robotics and human character animation has been approached in [15, 16]. Applications to humanoid robots control are also presented in [17, 18] with stability constraints. In [15], Kuffner splits the problem into two succesive tasks: path planning and path following. In this solution, a dynamic environment

is considered. A path is planned for the bounding cylinder of the character and a controller is used to synthesize an animation along this path. A collision invalidates the path and the whole process is started again. In [16], Raulo also considers dynamic environments. During the planning phase, a simplified representation of the environment is used. Then using a complete representation of the environment, the "Virtual Robot" tracks the planned trajectory, while information is gathered. The information allows to choose a strategy for modifying the animation and solving the possible collisions.
Our strategy differs by integrating the human motion control in the main loop of the path planning algorithm. This allows us to consider an animated path while testing its validity, without using some simplified bounding shapes (e.g a cylinder) for ensuring the collision avoidance.
Consequently, our contribution consists in synthesizing a steering method dedicated to the character animation and integrated in a randomized planner scheme. The architecture of the steering method can support several motion sub-controllers that act successively. The potential of the randomized motion planner algorithms on highly constrained problems benefits to our solution, respecting some low computing times.

In section 2, the model of the human character is introduced. Section 3 describes the first level of the steering method, while the core is detailed in section 4. Section 5 describes the integration and the implementation of the method in a randomized motion planner architecture.

2 Model and Motion Data

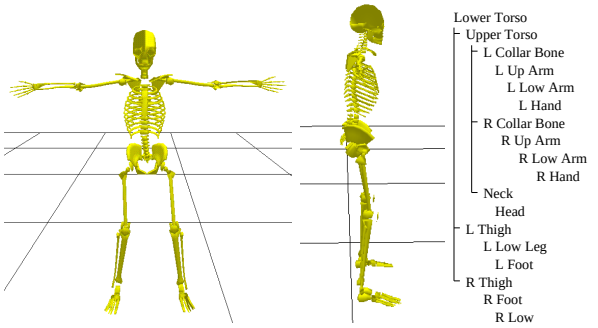

Figure 1: *Character's model*

Our virtual actor is modeled as a classic opened kinematic structure with 57 degrees of freedom (dof) detailed on figure 1. "Lower Torso" is the root of the kinematic structure. Position and orientation of the root gives us the situation of the character, noted $[x, y, z, \theta, \phi, \psi]$. The whole configuration of the character is then given by its situation attached to the angular values of each dof, and is denoted $[x, y, z, \theta, \phi, \psi, q_1, ..., q_n]$. Motion capture data depend on the model and represent the evolution of the configuration through the time, in several cases: walks, turns, runs, etc.

3 Trajectory of the root

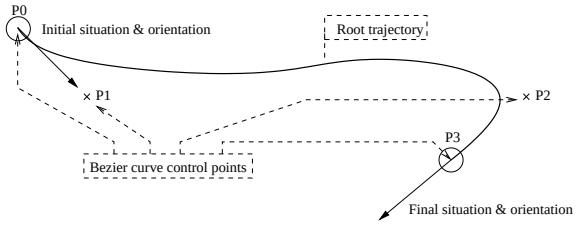

Figure 2: *Example of a trajectory*

The problem of the trajectory of the root (of the kinematic structure of the model) concerns only the evolution of the parameters $[x, y, \theta]$ and is the first level of our steering method. Inputs of this functionality are two configurations of the virtual actor's free configuration space ($\mathcal{C}_{free}$). The local human navigation is modeled as a third degree Bézier curve (see figure 2). As a result, the global path issued from the whole planning method is a composition of Bézier curves respecting C^1 continuity constraint, i.e. a B-Spline. The coordinates of the control points P_0 and P_3 are directly given by the inputs: x_{ini}, y_{ini}, x_{fin} and y_{fin}. P_1 and P_2 are computed with respect to θ_{ini}, θ_{fin} and a configurable distance D (which depends on the size of the character and on the distance between P_0 and P_3):

$$\begin{cases} P_1x = x_{ini} + Dcos(\theta_{ini}) \\ P_1y = y_{ini} + Dsin(\theta_{ini}) \\ P_2x = x_{fin} - Dcos(\theta_{fin}) \\ P_2y = y_{fin} - Dsin(\theta_{fin}) \end{cases} \quad (1)$$

The evolution of the parameters $[x, y, \theta]$ is given by some cartesian parametric equations:

$$\begin{cases} x(t) = \sum_{i=0}^{n} \frac{n!}{i!(n-i)} t^i (1-t)^{n-i} P_i x \\ y(t) = \sum_{i=0}^{n} \frac{n!}{i!(n-i)} t^i (1-t)^{n-i} P_i y \\ \theta(t) = arctan(y'(t)/x'(t)) \end{cases} \quad (2)$$

with $n = 3$, $0 \leq t \leq 1$ the parameter of the Bézier curve, and $P_i x$ the x coordinate for the control point number i, and so on. Other values are needed: arc length, tangential and rotation speeds. These can be analytically derived from equation 2.

4 Animation Module

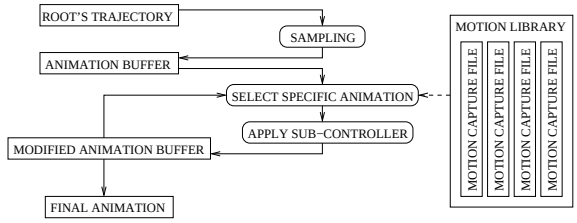

Figure 3: *Animation Module Structure*

The animation module is composed of two elements: a motion library and a set of specific sub-controllers. The successive application of the sub-controllers compose our human motion controller. The principle of the module is to sample the previously computed trajectory for the root. The animation buffer stores the samples. The content of the motion library is scanned, and with respect to the type of the motion data found, some specific sub-controllers are applied. The process is described on figure 3. The animation module is designed to be easily upgraded by adding new sub-controllers.

4.1 Motion Library

The motion library is a container for motion capture files. Each motion capture file is pre-processed: filtered (in the case of walk cycles) and characterized as explained in the next paragraph. The motion library contains different walk cycles, and other action descriptions: movements, postures, etc.

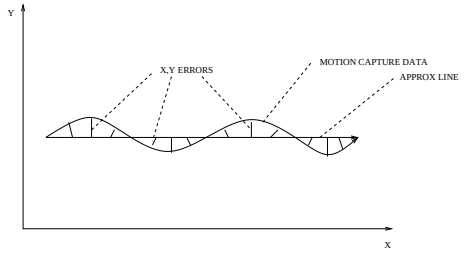

Figure 4: *Computing (x,y) errors around the computed linear approximation*

Preparing Walk Cycles. The walk sub-controller is based on motion capture blending ([8, 7, 9]). This method needs a preparation of motion data which have to be cyclic, with the same structure and parameterization.
The user is in charge of selecting some approximatively cyclic sequences into motion capture files, and of adding a description (see section 4.1).
The next phases of the preparation are automatically done during the planner's initialization phase. Most parameters of a motion capture are cyclic in the case of the walk. Only the situation's parameters are problematical as they are highly non-cyclic. We propose a filtering method in order to solve this problem. In the case of straight walks, the parameters of the position $x(t), y(t), z(t)$, given by the motion capture data, are approximated by a line whose analytical expression is given by a linear less squares fitting technique, defining $\tilde{x}(t), \tilde{y}(t), \tilde{z}(t)$ and average orientations. In the case of turning walks, the approximation shape is a circular arc. Some parameters derived from the approximation are kept: average tangential and rotation speeds.
Situation data $[x, y, z, \theta, \phi, \psi]$ are computed as errors around the approximation previously done. Errors are computed by $\epsilon_x = x(t) - \tilde{x}(t)$, etc. This computation is illustrated on figure 4 over both $(x(t), y(t))$ parameters. The set of parameters defining a motion is now denoted $[\epsilon_x, \epsilon_y, \epsilon_z, \epsilon_\theta, \epsilon_\phi, \epsilon_\psi, q_1, ..., q_n]$ and is almost cyclic. Fourier expansions are computed over these parameters, and a low-band filter is applied [11] in order to make sequences cyclic. Consequently, a set of parameters (α_k, β_k) characterizes each motion style. The evolution of a configuration parameter q_i can be computed using equation 3.

$$q_i(t) = \alpha_0 + \sum_{i=1}^{n} \alpha_i cos(2\pi ti) + \beta_i sin(2\pi ti) \qquad (3)$$

where $0 \leq t \leq 1$.

Characterizing Files. Adding a description to the motion data is crucial. As the previously described filtering method differs from a motion type to another, the user has to mention some characteristics. First, motion data are not limited to walk sequences; they can correspond to specific postures or movements. Also, some data do not concern all character's dof (eg. an arm-only movement description). The two following characteristics must be given:

- Animation type: for example walk (with sub-characteristic: straight or not), movement, posture, blocking task (the character must stop to execute the movement) etc.
- Animation chronological validity: the motion controller needs to know when the motion data must be taken into account. Walk cycles are generally always valid. But we could imagine that run cycles are valid only when the character is far from obstacles. Also, movement and postures are generally valid at a given period: at starts or ends of paths, etc.

4.2 Motion Controller

Sub-controllers. A sub-controller is an application rule dedicated to a data file type. The inputs

are the motion data, the validity periods and the animation buffer. The output is the modified animation buffer. The set of all sub-controllers present in the animation module constitutes our character's motion controller. The set of sub-controllers are applied successively with respect to the content of the animation library.

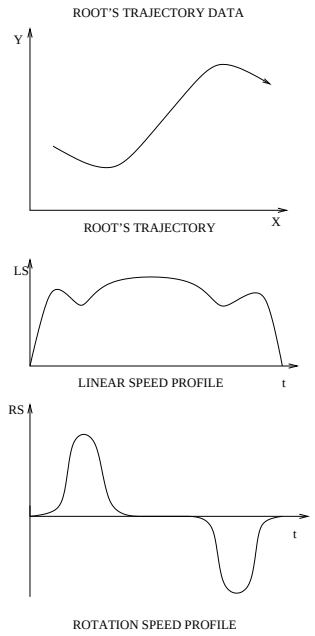

Figure 5: *Characteristics of the trajectory*

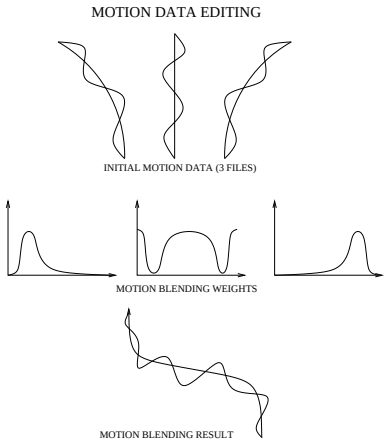

Figure 6: *Motion blending steps*

Walk controller The walk sub-controller synthesizes a walk animation around the previously computed trajectory of the root. Each frame of the animation buffer is scanned. Given the frame's characteristics (tangential and rotation speeds (v, ω)), the controller computes a locally valid set of parameter (α_b, β_b), thanks to the examples contained in the animation library and a motion blending technique. Then, a configuration is extracted from these parameters and projected on the situation of the character for the actual frame.

The algorithm of the controller is schematically illustrated on figure 6. For readability reasons, we illustrate the process over the parameters $[x, y]$ only. On figure 5, the parameters characterizing the trajectory of the root are given (see section 3 for equations). Figure 6 shows the initial motion data at the top (errors around approximated trajectory), and their evolution. The final projection over the trajectory is illustrated on figure 7.

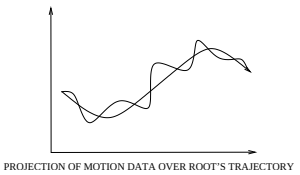

Figure 7: *Synthesized motion projection*

Generating a motion cycle with some desired tangential and rotation speeds (v_f, ω_f) is done by blending some examples of motion. Three motion cycles with the closest average speeds (denoted $(\omega_1, v_1), (\omega_2, v_2), (\omega_3, v_3)$) are selected in the animation library. The weights a, b, c of each motion respectively defined with (α_1, β_1), (α_2, β_2) and (α_3, β_3) can be computed by solving this linear system.

$$\begin{cases} av_1 + bv_2 + cv_3 = v_f \\ a\omega_1 + b\omega_2 + c\omega_3 = \omega_f \\ a + b + c = 1 \end{cases}$$

Finally, the desired motion parameters (α_b, β_b) are computed:

$$\begin{cases} \alpha_b = a\alpha_1 + b\alpha_2 + c\alpha_3 \\ \beta_b = a\beta_1 + b\beta_2 + c\beta_3 \end{cases} \quad (4)$$

A final step consists in finding the moment in the synthesized motion cycle to be used, in order to obtain a continuous motion all along the local path. The problem is solved by considering the distance between the two frames (the actual one and the previous one), the configuration of the character for the previous frame and the local tangential speed. Once calculated, an inverse Fourier transform is applied to determine all the value of each parameter of the configuration. Finally, the errors concerning the parameters $[x, y, \theta]$ are projected over the trajectory

and angular values of the other degrees of freedom are copied. The result is illustrated on figure 7.

Other Sub-controllers The animation module architecture is designed to integrate other sub-controllers. For example:

- A partial configuration evolution can be described in a motion capture data: arm-only movements for example. When the chronological validity (defined in subsection 4.1) is verified, contained motion data replace the animation buffer data on the concerned degrees of freedom.
- Another sub-controller allows to call the motion planner (with another steering method such as a linear one) in order to generate a transition between two configurations. This is how postures in the motion library are taken into account, and how a sequence of postures can be executed by the character.

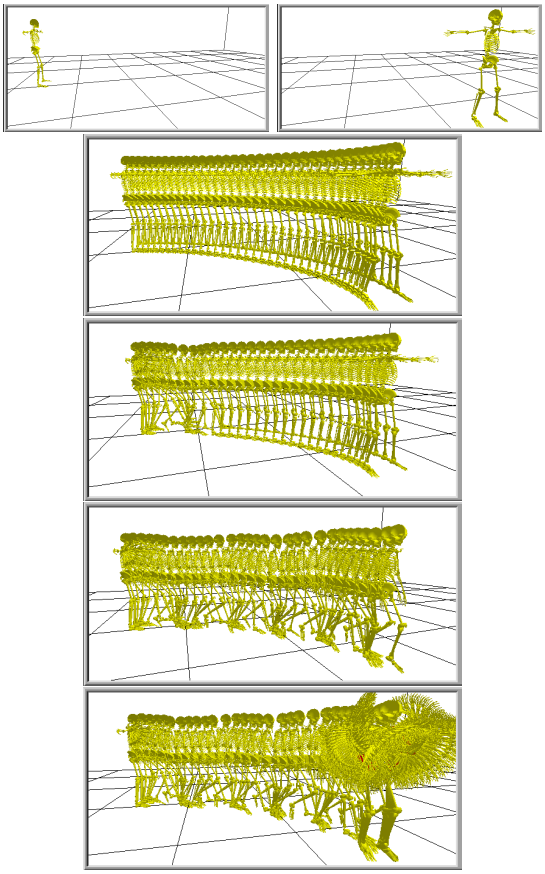

Figure 8: *Example of animation: a/ b/ Inputs, c/ Path, d/ Start Walk e/ Walk Control. f/ Macarena*

Evolution of the Animation. The evolution of the animation buffer, between each sub-controller application, is illustrated on figure 8. In this example, a sequence is first introduced at the beginning of the animation buffer: a walk start. Then, walk cycles are applied on the remaining part of the trajectory, processed by the walk controller. A transition is synthesized between the "start walk" sequence and the result of the walk controller. A linear interpolation between those two sources allows to smooth the result.
At last, some successive postures are executed by the character, which correspond to the macarena's dance steps. This is why at the bottom of the figure 8 the skeleton is moving its arms.

5 Walk planning

The steering method described in the two previous sections has been implemented within Move3D [19]. Move3D is a motion planning platform that integrates several randomized algorithms. Main components of such algorithms are:

- a steering method to compute admissible paths,
- a collision checker which is used both to select the nodes of the roadmap and to check whether an admissible path is collision-free or not.
- a roadmap builder in order to generate or extend the roadmap,
- a roadmap explorer in order to solve specific problems (which eventually calls the roadmap builder).

The global planner included into Move3D uses the visibility PRM technique [20] in order to reduce the size of the roadmap that captures the connectivity of the configuration space. Figure 9 shows an example of a collision free walk automatically computed by our planner. Considering the whole kinematic structure and the description of the environment in three dimensions allows us to find solutions in constrained environments. Figure 9 illustrates this potential, with a resulting trajectory close to the furniture (note that the skeleton's arm pass above the piano's stool). This result could not be obtained with a bounding cylinder around the human model.

6 Conclusion

We have presented a human walk planning method associating randomized motion planning and motion capture editing techniques. The first results obtained are promising. Several developments could improve realism, computing time, and application field. We are interested in investigating reactive planning methods to deal with dynamic environments. The character controller is also in permanent evolution. Also, we want to develop new sub controllers, especially for environment, objects or characters interaction problems.

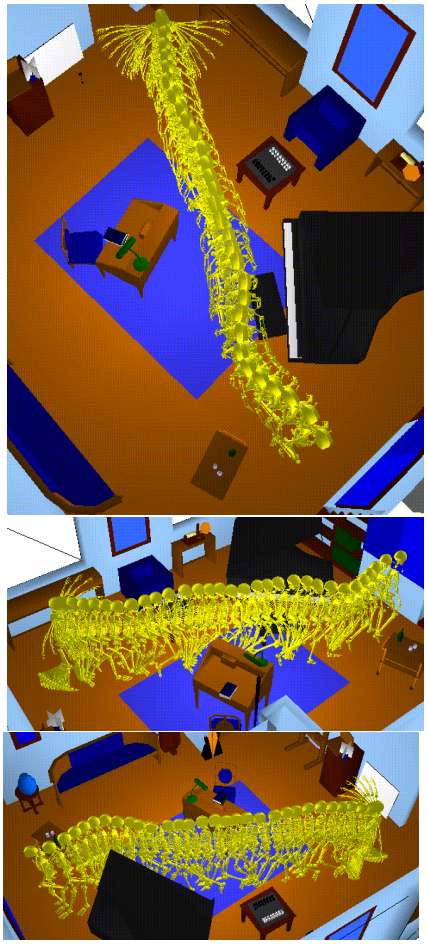

Figure 9: *A complete walk planning through a living room (3 different points of view: a/ b/ c/)*

Acknowledgment

Thanks to F. Forges from Ex-Machina who gave us the motion capture data used in our implementation.

References

[1] Rae Earnshaw, Nadia Magnetat-Thalmann, Demetri Terzopoulos, and Daniel Thalmann. Computer animation for virtual humans. *IEEE Computer Graphics and Applications*, pages 20–23, September/October 1998.

[2] Franck Multon, Laure France, Marie-Paule Cani, and Gilles Debunne. Computer animation of human walking: a survey. *The Journal of Visualization and Computer Animation*, 10:39–54, 1999. Published under the name Marie-Paule Cani-Gascuel.

[3] Jean-Claude Latombe. *Robot Motion Planning.* Boston: Kluwer Academic Publishers, Boston, 1991.

[4] Jean Paul Laumond. *Robot motion planning and control.* Springer-Verlag, 1993.

[5] R. Boulic, M. Thalmann, and D. Thalmann. A global human walking model with real-time kinematic personification. In *The visual computer*, pages 344–358, 1990.

[6] P. Faloutsos, M. van de Panne, and D. Terzopoulos. Composable controllers for physics-based character animation. In *Proc. ACM SIGGRAPH 2001 Conference, Los Angeles, CA*, 2001.

[7] A. Witkin and Z. Popovic. Motion warping. In *Proc. SIGGRAPH'95*, 1995.

[8] A. Bruderlin and L. Williams. Motion signal processing. In *Proc. SIGGRAPH'95*, 1995.

[9] Charles F. Rose. *Verbs and Adverbs : Multidimensionnal motion interpolation using radial basis functions.* PhD thesis, Princeton University, June 1999.

[10] Bobby Bodenheimer, Chuck Rose, Seth Rosenthal, and John Pella. The process of motion capture: Dealing with the data. *Computer Animation and Simulation '97, Eurographics*, pages 3–18, 1997.

[11] M. Unuma, K. Anjyo, and R. Takeuchi. Fourier principles for emotion-based human figure animation. In *Proc. od SIGGRAPH'95*, 1995.

[12] Shih kai Chung. *Interactively Responsive Animation of Human Walking in Virtual Environments.* PhD thesis, George Washington University, May 2000.

[13] Lydia Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. Technical Report CS-TR-94-1519, 1994.

[14] L. Kavraki and J. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA), pp. 2138-2145, 1994.*, 1994.

[15] James J. Kuffner Jr. Goal-directed navigation for animated characters using real-time path planning and control. In *CAPTECH*, pages 171–186, 1998.

[16] D. Raulo, J.M. Ahuactzin, and C. Laugier. Controlling virtual autonomous entities in dynamic environments using an appropriate sense-plan-control paradigm. In *Proc. of the 2000 IEEE/RS. International Conference on Intelligent Robots and Systems*, 2000.

[17] James Kuffner, Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue. Graphical simulation and high-level control of humanoid robots. *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'00)*, 2000.

[18] James Kuffner, Masayuki Inaba, and Hirochika Inoue. Automating object manipulation tasks for humanoid robots. *Proc. of 1st Int. Conf on Robotics and Mechatronics (ROBOMEC'00)*, pages 2P2–79–103, 2000.

[19] T. Siméon, JP. Laumond, and F. Lamiraux. Move3d: a generic platform for motion planning. In *4th Internation Symposium on Assembly and Task Planning, Japan.*, 2001.

[20] T. Siméon, J.P. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. In *Advanced Robotics Journal 14(6)*, 2000.