GPU-accelerated Real-Time 3D Tracking for Humanoid Autonomy

Philipp Michel[†], Joel Chestnutt[†], Satoshi Kagami[‡], Koichi Nishiwaki[‡],
James Kuffner[†‡] and Takeo Kanade[†‡]

[†]The Robotics Institute
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
{pmichel,chestnutt,kuffner,kanade}@cs.cmu.edu

[‡]Digital Human Research Center
National Institute of Advanced Industrial Science and Technology
2-41-6, Aomi, Koto-ku, Tokyo
135-0064, Japan
{s.kagami,k.nishiwaki}@aist.go.jp

We have accelerated a robust model-based 3D tracking system by programmable graphics hardware to run online at frame-rate during operation of a humanoid robot and to efficiently auto-initialize. The tracker recovers the full 6 degree-of-freedom pose of viewable objects relative to the robot. Leveraging the computational resources of the GPU for perception has enabled us to increase our tracker's robustness to the significant camera displacement and camera shake typically encountered during humanoid navigation. We have combined our approach with a footstep planner and a controller capable of adaptively adjusting the height of swing leg trajectories. The resulting integrated perception-planning-action system has allowed an HRP-2 humanoid robot to successfully and rapidly localize, approach and climb stairs, as well as to avoid obstacles during walking.

Key Words: GPU, Tracking, Perception, Planning, Humanoid Autonomy

1. Introduction

Perception on humanoid robots presents several unique challenges. Approaches to localization and mapping must deliver accurate results to comply with the small error tolerances imposed by the walking controller if the robot is to successfully step onto surfaces or avoid obstacles. Moreover, they must be able to deal with rapid scene changes, large camera displacement and camera shakiness and should operate in real-time, since pausing for deliberation or sensing is often not an option. However, the complexity of vision processing often implies that these requirements cannot all be met at once with the traditional CPU-based computational resources available. In this paper, we present a GPU implementation of a model-based 3D tracking algorithm which we have applied specifically to the problem of humanoid locomotion. Our system robustly fits the visible model edges of a given object to edge features extracted from the video stream, yielding the full 6DOF pose of the object relative to the camera. The recovered pose, together with the robot kinematics, allows us to accurately localize the robot with respect to the object and to generate environment maps. These can then be used to plan a sequence of footsteps that, when executed, allow the robot to quickly and successfully circumnavigate obstacles and climb stairs.

2. Related Work

A large body of work exists relating to model-based 3D object tracking and associated visual servoing approaches. For a more complete overview, please refer to Lepetit & Fua's excellent survey [1]. Early work by Gennery [2] first focused on tracking objects of known 3D structure, with Lowe [3] pioneering the fitting of model edges to image edges. Harris' RAPiD tracker [4] first achieved such fitting in real-time, with a range of improvements to the original algorithm having been proposed [5]–[7]. Other approaches employ appearance-based methods to perform tracking [8] or view tracking as a combination of wide-baseline matching and bundle adjustment relying on so-called keyframe information gathered offline [9]. There is an ever increasing body of work regarding the use of GPUs for general purpose computation. Several good overview resources exist [10], [11]. Particularly relevant to perception is the work by Fung et. al. [12]. The demands of using a locomoting humanoid as the perception platform have implied that several perception approaches restrict their operation to reactive obstacle detection and avoidance [13]. Others have restricted the recovered environment information to 2D occupancy grids [14], have employed external sensors to aid in robot localization and map building [15] or use stereo for reconstruction [16].

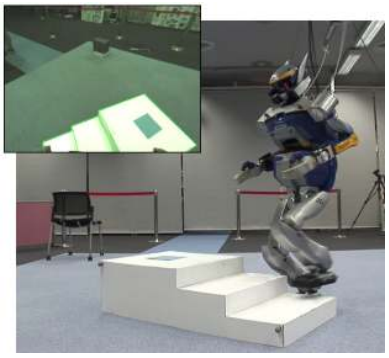


Fig. 1. The HRP-2 humanoid autonomously climbing a set of stairs. Environment mapping and robot localization is accomplished online using our GPU-accelerated 3D tracker (tracker view inset).

3. Model-based 3D Tracking & Pose Recovery

3.1 Overview

Our approach to monocular model-based 3D tracking closely follows the method proposed by Drummond and Cipolla [17]. The reader is referred to [18] for a more thorough explanation. We initialize and subsequently update an estimate of the matrix representing the $SE(3)$ pose of the tracked object relative to the camera. This 3×4 matrix $\mathbf{E}$ corresponds to the extrinsic camera matrix and transforms points from world coordinates to camera coordinates. We also gather a 3×3 matrix of intrinsic parameters $\mathbf{K}$, during an offline calibration step. Together, these matrices form the camera projection matrix $\mathbf{P} = \mathbf{KE}$.

To estimate the relative pose change between two consecutive frames, we project the object model onto the image plane according to the latest estimate of the pose $\mathbf{E}_t$ and initialize a set of regularly spaced so-called control nodes along those projected edges. We then use these control nodes to match the visible projected model edges to edge features extracted from the camera image using a Canny edge detector [19]. The errors in this matching can then be used to find an update $\Delta\mathbf{E}$ to the extrinsic parameter matrix using robust linear regression. The updated pose of the object is finally calculated as $\mathbf{E}_{t+1} = \mathbf{E}_t\Delta\mathbf{E}$ and the procedure repeated for the next frame.

3.2 Model-based 3D object tracking

The recovery of the inter-frame pose update $\Delta\mathbf{E}$ can be implemented by considering the set of control nodes along the visible model edges and, for each, determining the perpendicular distance to the closest image edge using a one-dimensional search.

The camera projection matrix takes a point from world coordinates to projective camera coordinates via $(u, v, w)^T = \mathbf{P}\ (x, y, z, 1)^T$, with pixel coordinates given by $x = u/w$ and $y = v/w$. To recover the rigid transform $\Delta\mathbf{E}$, we consider the six generating motions which comprise it, namely translations in the x, y and z directions and rotations about the x, y and z axes, represented by the 4×4 matrices $\mathbf{G}_1$ to $\mathbf{G}_6$. These generating motions form a basis for the vector space of derivatives of $SE(3)$ at the identity and can be considered velocity basis matrices.

The pose update $\Delta\mathbf{E}$ can be constructed from these Euclidean generating motions via the exponential map as $\Delta\mathbf{E} = \exp\left(\sum_{i=1}^{6} \mu_i \mathbf{G}_i\right)$. The motion vector $\boldsymbol{\mu}$ thus parameterizes $\Delta\mathbf{E}$ in terms of the six generating motions $\mathbf{G}_1$ to $\mathbf{G}_6$. It is $\boldsymbol{\mu}$ that we recover using robust linear regression.

If a particular control node ξ with homogeneous world coordinates $\mathbf{p}^\xi = (x, y, z, 1)$ is subjected to the i^{th} generating motion, the resulting motion in projective image coordinates is given by $(u', v', w')^T = \mathbf{P}\mathbf{G}_i\,\mathbf{p}^\xi$. This can be converted to pixel coordinates as follows:

$$\mathbf{L}_i^\xi = \begin{pmatrix} \tilde{u}' \\ \tilde{v}' \end{pmatrix} = \begin{pmatrix} \frac{u'}{w} - \frac{uw'}{w^2} \\ \frac{v'}{w} - \frac{vw'}{w^2} \end{pmatrix}$$

We can project this motion onto the model edge normal $\hat{\mathbf{n}}$ at the control node as $f_i^\xi = \mathbf{L}_i^\xi \cdot \hat{\mathbf{n}}$.

Suppose we have determined during our 1D edge search along the model edge normal that control node ξ is at a distance d^ξ from the closest image edge extracted from the video frame. Considering the set of control nodes in its entirety, we can calculate the motion vector $\boldsymbol{\mu}$ by fitting d^ξ to f_i^ξ for each control node via the usual least-squares approach:

$$g_i = \sum_\xi d^\xi f_i^\xi;\ C_{ij} = \sum_\xi f_i^\xi f_j^\xi;\ \mu_i = \sum_j C_{ij}^{-1} g_j$$

We can now use the recovered motion vector $\boldsymbol{\mu}$ to reconstruct the inter-frame pose update via the exponential map.

3.3 Robust fitting / Pose filtering

The standard least-squares fitting method outlined above is well known to be adversely influenced by the presence of outliers, which are particularly present when dealing with rapidly changing and often cluttered views of the world from the robot camera. We thus employ Iteratively Reweighted Least Squares (IRLS) for robust fitting. The residuals from an initial ordinary least squares fitting step are subsequently re-weighted according to Tukey's biweight, giving lower weights to points that do not fit well. We iterate the reweighted fitting a fixed number n of times until the residuals change only marginally (for the experiments in this paper, $n = 5$).

Then, still considering the *same* two adjacent frames in the video, we re-project the control nodes using the pose $\mathbf{E}_t\Delta\tilde{\mathbf{E}}$ and re-start the entire inter-frame tracking process, including edge search and IRLS fitting. Iterating essentially the whole pose update process in this way for a single pair of frames ensures the most accurate and robust model-to-edge fitting, but is computationally very expensive. However, leveraging the GPU for all of the image processing and edge search leaves us with enough CPU resources to execute the model fitting process several iterations for each frame, thus significantly increasing robustness over a CPU-only implementation.

To further increase the robustness of the tracker against incorrect snapping to strong misleading background contours, we consider multiple edge hypotheses for each control node during the fitting stage. For each control node ξ, we search along the model edge normal and record distance measurements d_k^ξ to the k closest image edges found, rather than merely attributing a single measurement to each control node. During the initial fitting step, we take *all* hypotheses extracted for *all* control nodes into account with equal weight. During the subsequent IRLS fitting process, weights are computed for each hypothesis at every point. Now, at each control node, only the residual corresponding to the hypothesis with the *lowest* weight contributes to the fit at each iteration.

We combine measurements (i.e. the recovered pose from each inter-frame tracking step, $\mathbf{E}_t$) using a Discrete Extended Kalman Filter [20]. The filter maintains a 12 dimensional internal state, representing both pose (6DOF) and velocity (6DOF) of the object being tracked. These are stored as an $SE(3)$ pose matrix and a velocity 6 vector, holding translational and rotational velocities. The filter aids in appropriately integrating successive pose measurements to eliminate jitter and provide 'smoother' pose recovery over time. The filter's embedded dynamics model also provides an estimate of how the object being tracked is expected to move in the near future, and proves very helpful when tracking rapidly moving objects. We use the filter state after each prediction step to start our inter-frame pose recovery process.

4. GPU-based implementation

All aspects of our 3D tracking system involving image or geometry processing are either executed entirely in the GPU's fragment shaders or involve hardware-accelerated OpenGL. We have implemented our method using a cascade of fragment programs, shown in Figure 2, written using NVIDIA's Cg language and operating on image data stored as textures in GPU memory. The latest incoming video frame serves as input to the filter cascade, which ultimately outputs a texture containing the edge-normal distances d^ξ for all control nodes on the visible projected model edges of the object. All steps in between operate on data stored locally on the GPU as the output of a previous step.

We use a firewire camera, standalone or mounted on the robot head, to gather video at resolutions of up to 1024×768 pixels and 30 frames per second. We perform YUV-RGB color conversion, radial undistortion of the camera image according to the recovered calibration parameters and Canny edge detection on the GPU. This results in a single rectified binary image texture indicating presence or absence of an edge at each pixel.

4.1 Model projection & edge search

To fit edges rendered using the current pose estimate to image edges, we assume the existence of a simple 3D model of the object, easily generated using CAD or photogrammetry software. In particular, we use Google Sketchup and its Photo Match feature to quickly generate geometrically accurate textured models from a few photographs. We then render our model onto the image plane according to the latest pose estimate, performing hidden line removal efficiently using depth-buffered OpenGL, resulting in a binary texture containing only the visible edges of the model.

We initialize a number of control nodes along the model edges, spaced evenly in image coordinates. Control node information is provided to the edge search fragment program as a single four channel RGBA texture, with the red channel indicating presence/absence, the green and blue channels encoding the x and y components of model edge normal at the control node, and their signs being integer-encoded in the alpha channel.

The edge search fragment program then steps along the true model edge normal (albeit quantized to pixel coordinates) trying to detect the $k = 4$ closest image edges to

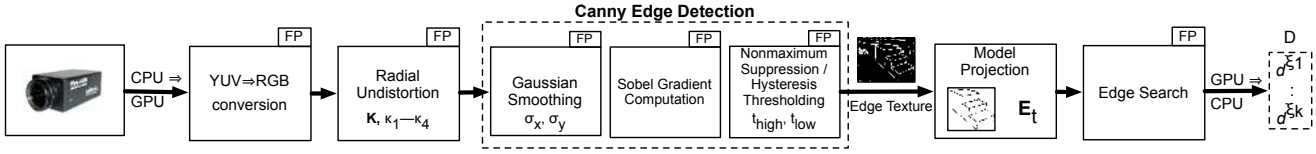


Fig. 2. GPU fragment program cascade defining flow of image processing, model projection and edge search.

the control node in either the positive or negative normal direction. Search is performed up to a certain cutoff distance. If no image edges are found within that distance, the control node is ignored and does not contribute to the solution fit.

Although the search distance, the number of hypotheses and the number of control points search is performed on directly affect the running time of the edge search process, we have not been able to saturate our GPU-implementation even with many hundreds of control nodes and edge search distances spanning more than 50 pixels in either direction of the normal. Furthermore, the GPU's abundant compute resources have also enabled us to handle the tracking of multiple objects present in the scene in a straightforward manner. A separate pose estimate is maintained throughout the tracking process for each object of interest, with a *single* texture containing control node information for *all* objects to the edge search fragment program. During the fitting stage, search results are then associated with their respective objects and fitting proceeds separately for each one.

4.2 Tracker Initialization

Many previous approaches to edge-based 3D tracking rely on an a-priori step of manual initialization to establish the initial pose of the object $\mathbf{E}_0$. We have implemented an automatic initialization method that rapidly establishes 2D-3D point correspondences, from which the initial pose can be recovered. It relies on a textured 3D model of the object being tracked, which we render from a variety of viewpoints (sampled uniformly or from a given set of viewpoints we are likely to encounter during operation). The resulting model images are stored together with the pose from which they were rendered.

We use features based on David Lowe's Scale Invariant Feature Transform [21] to perform matching between incoming camera images and our database of model images. SIFT features are extracted from each of the model images and from incoming camera images very rapidly on the GPU using a modified version of SiftGPU [22]. Extracting about 500 features from an image takes roughly 80ms on the GPU, compared to around 6 seconds for a typical CPU implementation. We then match the input image features to each of the model images using a Best-Bin-First KD-tree search and RANSAC to yield a set of inliers. The model image with the largest number of inliers is chosen.

Given these 2D-2D matches and the 3D model of the object of interest, we are able to recover the 3D coordinates of the SIFT keypoints in the model images. We use OpenGL's `gluUnproject()` function to very efficiently determine the 3D object coordinates of a 2D point using the graphics hardware. The resulting set of 2D-3D matches (associating keypoints in the input images with 3D points on the surface of the object model via one of the model images) is then used to find the initial pose using the POSIT algorithm [23].

5. Robot Localization / Environment Mapping / Planning

To localize the robot, we establish a map coordinate system in which the object being tracked is assumed to remain at a fixed location, given by a transform ${}^m_oT$. Once we have recovered the pose of the object in camera coordinates (given, say, by a transform ${}^c_oT$), it is easy to position the camera relative to the object. The pose of the camera in map coordinates is then straightforwardly recovered as ${}^m_cT = {}^m_oT\,{}^o_cT = {}^m_oT\,({}^c_oT)^{-1}$, essentially positioning the camera in a consistent coordinate system relative to the object of interest. For planning, we require the precise location of the robot's feet. We recover this using the robot kinematics, which supplies another transform, ${}^c_fT$, locating the robot foot relative to the camera at any instant in time. When chained with ${}^m_cT$, this locates the foot in map coordinates. From the known shape of the object being tracked and its fixed position in map coordinates, we can easily generate a height map describing the robot environment by rendering it top-down using an orthographic projection.

The navigation planning performed for the experiments in this paper uses a modified version of our previously described footstep planner [24]. The planner reduces the problem of motion planning for the humanoid to planning a sequence of footsteps for the robot to follow, along with swing leg trajectories and step timings that move the robot's legs from foothold to foothold. Using this information, a walking controller then generates a dynamically stable motion to walk along the desired path.

6. Results

6.1 Standalone tracker operation

To establish the operational performance of our tracker, we first used a standalone firewire camera attached to a commodity PC equipped with an NVIDIA GeForce 8800GTX PCI-Express GPU. The system tracked a set of white stairs at 30fps while an experimenter moved the handheld camera around freely. Compared to manual measurements, the recovered translation from the camera to the object was accurate to within 1cm at a camera-object distance range of 1–2m. Figure 3(a) shows a typical view with the tracked model superimposed in green, Figure 3(b) shows a view of the model superimposed on the extracted image edges during object occlusion with a checkerboard. Figure 3(c) shows a view of the tracker during severe occlusion by an experimenter walking in front of the camera.

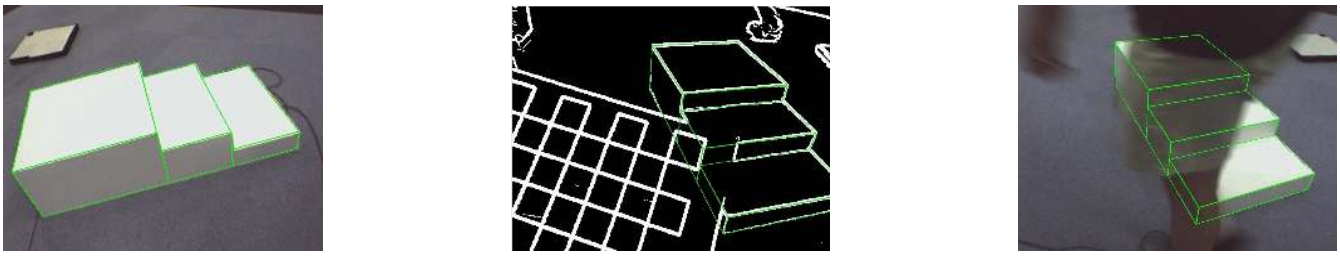
6.2 Robot experiments

Our robot experiments combine the GPU-accelerated 3D tracking system, footstep planner, and walking and balance controller operating on-line on an HRP-2 humanoid robot. The tracker processes live video supplied by a robot head-mounted camera to an off-board computer, again tracking a set of stairs in the environment, which the robot climbs or avoids in our experiments.

We carried out 15 stair climbing experiments with the robot starting from a wide variety of distances from and orientations relative to the stairs, during 13 of which HRP-2 successfully reached the top of the stairs. The average length of a successful climbing sequence from the point the robot started moving was under 8 seconds. Figure 4(a) shows HRP-2 successfully approaching and climbing our set of stairs. Figure 4(b) shows HRP-2 navigating around the same set of stairs.

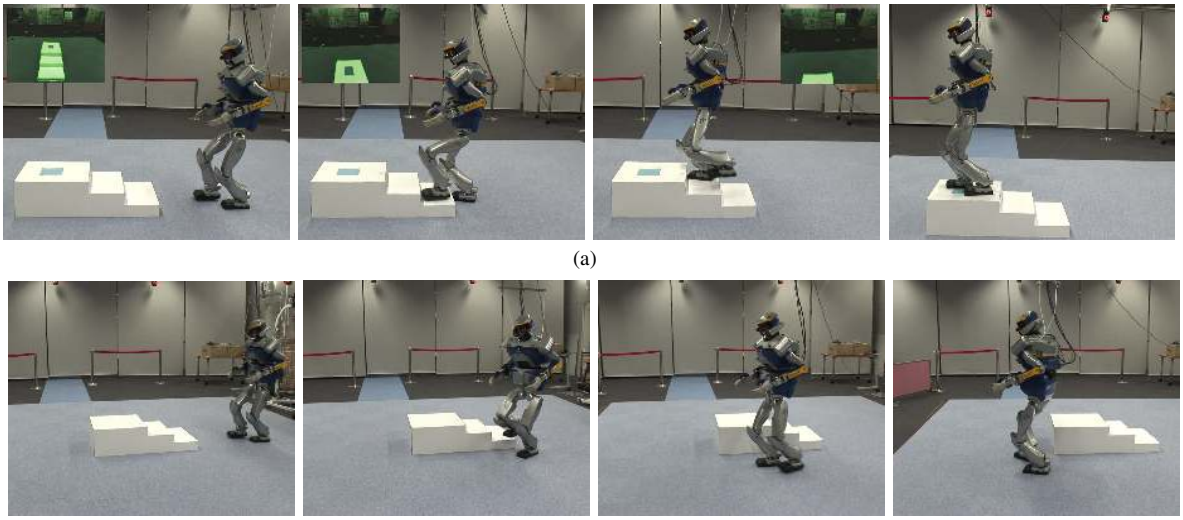
7. Discussion

We have presented a fully-integrated online perception-planning-execution system for a humanoid robot employing a GPU-accelerated model-based 3D tracker for perception. The increased robustness afforded by leveraging the GPU has enabled an HRP-2 humanoid to successfully accomplish



(a) (b) (c)

Fig. 3. Stairs being tracked during handheld camera sequence *(a)*. View of model-edge to image-edge fitting during occlusion *(b)*. Tracker operation under severe occlusion *(c)*.



(a)

(b)

Fig. 4. Examples of GPU-accelerated tracking used for mapping and localization during humanoid locomotion: HRP-2 autonomously climbing *(a)* and avoiding *(b)* a set of stairs. Insets in top row show tracker view during execution. Stairs are no longer visible from the top step in the rightmost image of *(a)*.

complex locomotion tasks such as stair climbing and obstacle avoidance with a speed and flexibility not achieved before.

As future research, we have been working on exploiting our tracker for other humanoid tasks such as visual servoing for grasping. We have also been investigating a tighter coupling between the perception and planning stages of our system by having the planning stage reason explicitly about perception. We believe that GPUs will play an increasingly important role as an implementation platform for robotic perception algorithms, enabling humanoid robots to autonomously perform increasingly complex tasks in everyday, real-world environments.

References

[1] V. Lepetit and P. Fua, "Monocular model-based 3d tracking of rigid objects," *Found. Trends. Comput. Graph. Vis.*, vol. 1, no. 1, pp. 1–89, 2006.

[2] D. B. Gennery, "Visual tracking of known three-dimensional objects," *Int. J. Comput. Vision*, vol. 7, no. 3, pp. 243–270, 1992.

[3] D. G. Lowe, "Robust model-based motion tracking through the integration of search and estimation," *Int. J. Comput. Vision*, vol. 8, no. 2, pp. 113–122, 1992.

[4] C. Harris, "Tracking with rigid models," *Active vision*, pp. 59–73, 1993.

[5] M. Armstrong and A. Zisserman, "Robust object tracking," in *Proc. Asian Conference on Computer Vision*, 1995, pp. 58–61.

[6] T. Drummond and R. Cipolla, "Real-time tracking of multiple articulated structures in multiple views," in *ECCV '00: Proceedings of the 6th European Conference on Computer Vision-Part II*. London, UK: Springer-Verlag, 2000, pp. 20–36.

[7] A. I. Comport, E. Marchand, and F. Chaumette, "A real-time tracker for markerless augmented reality," in *ACM/IEEE Int. Symp. on Mixed and Augmented Reality, ISMAR'03*, Tokyo, Japan, October 2003, pp. 36–45.

[8] F. Jurie and M. Dhome, "Real time tracking of 3d objects : an efficient and robust approach." *Pattern Recognition*, vol. 35, no. 2, pp. 317–328, 2002.

[9] L. Vacchetti, V. Lepetit, and P. Fua, "Stable real-time 3d tracking using online and offline information," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 26, no. 10, pp. 1385–1391, 2004.

[10] M. Pharr and R. Fernando, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, March 2005.

[11] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," in *Eurographics 2005, State of the Art Reports*, Aug 2005, pp. 21–51.

[12] J. Fung and S. Mann, "Openvidia: parallel gpu computer vision," in *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, 2005, pp. 849–852.

[13] M. Yagi and V. Lumelsky, "Local on-line planning in biped robot locomotion amongst unknown obstacles," *Robotica*, vol. 18, no. 4, pp. 389–402, 2000.

[14] P. Michel, J. Chestnutt, J. Kuffner, and T. Kanade, "Vision-guided humanoid footstep planning for dynamic environments," in *Proc. of the IEEE-RAS/RSJ Int. Conf. on Humanoid Robots (Humanoids'05)*, December 2005, pp. 13–18.

[15] P. Michel, J. Chestnutt, S. Kagami, K. Nishiwaki, J. Kuffner, and T. Kanade, "Online environment reconstruction for biped navigation," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'06)*, Orlando, FL, USA, May 2006.

[16] K. Sabe, M. Fukuchi, J.-S. Gutmann, T. Ohashi, K. Kawamoto, , and T. Yoshigahara, "Obstacle avoidance and path planning for humanoid robots using stereo vision," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'04)*, New Orleans, LA, USA, April 2004.

[17] T. Drummond and R. Cipolla, "Real-time tracking of complex structures with on-line camera calibration," in *Proc. of the British Machine Vision Conference (BMVC'99)*, Nottingham, UK, September 1999.

[18] P. Michel, J. Chestnutt, S. Kagami, K. Nishiwaki, J. Kuffner, and T. Kanade, "Gpu-accelerated real-time 3d tracking for humanoid locomotion and stair climbing," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'07)*, October 2007, pp. 463–469.

[19] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–698, November 1986.

[20] G. Welch and G. Bishop, "An introduction to the kalman filter," University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, Tech. Rep., 1995.

[21] D. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[22] C. Wu, "SiftGPU," Web: http://cs.unc.edu/~ccwu/siftgpu/.

[23] D. DeMenthon and L. Davis, "Model-based object pose in 25 lines of code," in *ECCV '92: Proceedings of the Second European Conference on Computer Vision*. London, UK: Springer-Verlag, 1992, pp. 335–343.

[24] J. Chestnutt, J. Kuffner, K. Nishiwaki, and S. Kagami, "Planning biped navigation strategies in complex environments," in *Proc. of the IEEE-RAS/RSJ Int. Conf. on Humanoid Robots (Humanoids'03)*, Munich, Germany, October 2003.