Proceedings of the 2007 IEEE/RSJ International
Conference on Intelligent Robots and Systems
San Diego, CA, USA, Oct 29 - Nov 2, 2007

GPU-accelerated Real-Time 3D Tracking for Humanoid Locomotion and Stair Climbing

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

***Abstract*— For humanoid robots to fully realize their biped potential in a three-dimensional world and step over, around or onto obstacles such as stairs, appropriate and efficient approaches to execution, planning and perception are required. To this end, we have accelerated a robust model-based three-dimensional tracking system by programmable graphics hardware to operate online at frame-rate during locomotion of a humanoid robot. 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.**

I. Introduction

Research developments in humanoid robotics have led to a series of legged robots that exhibit impressive locomotion skills in cluttered, three-dimensional, inclined or uneven terrain. Lest they be treated as mere planar mobile robots, however, the unique walking abilities of humanoids need to be taken into account during all stages of autonomous navigation, from sensing through planning to execution. In the case of perception, this presents several concrete challenges. Approaches to robot localization and environment 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, rapid scene changes, large camera displacement and camera shakiness occur naturally with quickly moving highly articulated humanoids, and must be handled by the sensor system. Also, unlike for wheeled robots, pausing movement for deliberation or to gather sensor readings is usually not an option—perception must operate in real-time. Unfortunately, the complexity of vision processing often implies that these requirements cannot all be met at once with the computational resources available.

Graphics Processing Units (GPUs) have of late gained considerable popularity as cheap, powerful and programmable general purpose processors outside their original application domain. With recent models able to sustain over 150 GFLOPS and due to their highly parallel architecture, GPUs make very suitable implementation platforms for perception algorithms.

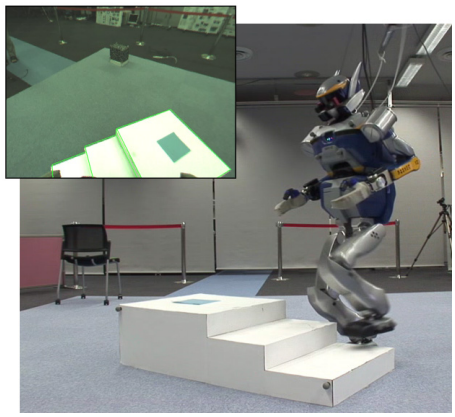


Fig. 1. The HRP-2 humanoid autonomously climbing a set of stairs. Environment mapping and robot localization is accomplished online using our GPU-accelerated 3D tracker (tracker view inset).

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 outlined above. We believe that mandating the existence of even a simple 3D model for objects of interest is not an unreasonable requirement considering the well structured indoor operating environments currently typical for humanoids. Our system employs a successful prior algorithm to robustly fit a series of control nodes initialized from 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 model is then reprojected using the recovered pose, and the fitting process repeated for the next frame. The recovered pose, together with the robot kinematics, allows us to accurately localize the robot with respect to the object and to generate height-based maps of the robot environment. These can then be used to plan a sequence of footsteps that, when executed, allow the robot to circumnavigate obstacles and climb stairs with a speed, flexibility and success rate not achieved before.

The remainder of this paper is organized as follows. We review some related research in Section II. The detailed operation of our tracking system and its implementation on the GPU is presented in Section III and Section IV, respectively. In Section V, we describe how the output of the tracker is used to localize the robot and to generate environment maps for robot navigation. The height-based planning and execution components of our robot implementation are the subject of Section VI. Section VII presents experimental re-

1-4244-0912-8/07/$25.00 ©2007 IEEE.

sults for our tracking system and its application to humanoid locomotion. Finally, we summarize in Section VIII, discuss current limitations and outline future work.

II. 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], [4] pioneering the fitting of model edges to image edges. Harris' RAPiD tracker [5] first achieved such fitting in real-time, with a range of improvements to the original algorithm having been proposed [6], causing edge- and contour-based tracking systems to maintain significant popularity [7]–[10]. Other approaches employ appearance-based methods to perform tracking [11] or view tracking as a combination of wide-baseline matching and bundle adjustment relying on so-called keyframe information gathered offline [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], [14]. Others have restricted the recovered environment information to 2D occupancy grids [15], have employed external sensors to aid in robot localization and map building [16] or use stereo for reconstruction [17].

Several bipeds have successfully accomplished stair climbing. Sony's QRIO robot uses stereo to reconstruct stairs and climb them gradually, step-by-step [18]. The Johnnie walking robot has successfully and quickly climbed stairs detected reactively using vision [19], [20]. Honda's ASIMO humanoid [21] first positions itself precisely with respect to a set of stairs equipped with fiducials and then executes fixed footstep sequence, adjusted according to the contact force with each step, to climb them. The H7 humanoid [22] has also successfully climbed a set of stairs positioned in front of it. Most prior stair-climbing approaches carry some a-priori knowledge of the shape or location of the stairs, or alternatively tend to carefully execute each stepping-up motion in isolation, foregoing continuous execution and smoothness for safety.

There is an ever increasing body of work regarding the use of GPUs for general purpose computation. Several good overview resources exist [23], [24]. Particularly relevant to perception is the work by Fung et. al. [25].

III. Model-based 3D Tracking & Pose Recovery

A. Overview

Our approach to monocular model-based 3D tracking closely follows the method proposed by Drummond and Cipolla [26]. We manually initialize and subsequently maintain and recursively 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 parameter matrix of the camera and transforms points from world coordinates to camera coordinates. Together with the 3×3 matrix of intrinsic parameters $\mathbf{K}$, which we gather during a one-off calibration step using Zhang's camera calibration method [27], it forms the camera projection matrix $\mathbf{P} = \mathbf{KE}$.

The pose estimation process then operates as follows. 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 [28]. 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.

B. Model-based 3D object tracking

In this section, we summarize how the recovery of the inter-frame pose update outlined above 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.

Recall that 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 $\Delta\mathbf{E}$ representing the inter-frame pose update, 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. These generating motions form a basis for the vector space of derivatives of $SE(3)$ at the identity and can be represented by the following six matrices, which can be considered velocity basis matrices:

$$\mathbf{G}_1=\begin{bmatrix}0&0&0&1\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix},\quad \mathbf{G}_2=\begin{bmatrix}0&0&0&0\\0&0&0&1\\0&0&0&0\\0&0&0&0\end{bmatrix},\quad \mathbf{G}_3=\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&1\\0&0&0&0\end{bmatrix},$$

$$\mathbf{G}_4=\begin{bmatrix}0&0&0&0\\0&0&-1&0\\0&1&0&0\\0&0&0&0\end{bmatrix},\quad \mathbf{G}_5=\begin{bmatrix}0&0&1&0\\0&0&0&0\\-1&0&0&0\\0&0&0&0\end{bmatrix},\quad \mathbf{G}_6=\begin{bmatrix}0&-1&0&0\\1&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}$$

The pose update $\Delta\mathbf{E}$ can be constructed from these Euclidean generating motions via the exponential map [29] 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 will 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:

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

Algorithm 1: IRLS(D, F)
Perform iteratively reweighted least squares model-edge to image-edge fitting

Data: $D = \{d^{\xi_1}, \ldots, d^{\xi_k}\}$: *measured* edge-normal distances to closest image edge for all k control nodes
$F = \{f^{\xi_1}_{\{1\ldots6\}}, \ldots, f^{\xi_k}_{\{1\ldots6\}}\}$: edge-normal motion resulting from each of the 6 generators for all k control nodes
Result: $\boldsymbol{\mu}$: motion vector parameterizing pose update

$\boldsymbol{\mu}$ = ORDINARYLEASTSQUARES(D, F);
while $iteration < max\_iterations$ **do**
  for *each control node* ξ **do**
    $residual^{\xi}$ = d^{ξ} - $\sum_{i=1}^{6} f_i^{\xi}\mu_i$;
  end
  MAD = MEDIANABSOLUTEDEVIATION($residuals$);
  for *each control node* ξ **do**
    $c = \frac{abs(residual^{\xi}) \times 0.6745}{4.685 \times MAD}$;
    $weight^{\xi} = \begin{cases} (1-c^2)^2 & \text{if } c < 1 \\ 0 & \text{otherwise} \end{cases}$
  end
  $\boldsymbol{\mu}$ = WEIGHTEDLEASTSQUARES($D, F, weights$);
end

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

C. Robust linear regression

The standard least-squares fitting method outlined above is well known to be adversely influenced by the presence of outliers. Since we are dealing with rapidly changing and often cluttered views of the world from the robot camera, such outliers occur universally. For instance, when there is a weak edge response from the object being tracked, but a strong edge resulting from other scene structure close to the object-background boundary in the image, the 1D edge search process will falsely associate one or several control nodes with the background edge. A variety of robust estimation methods have been proposed to ameliorate this problem, including RANSAC and M-estimators [30].

We employ Iteratively Reweighted Least Squares (IRLS) for robust fitting. The first iteration consists of a step of ordinary least squares with equally weighted measurements and with the edge-normal distances d^{ξ} forming the residuals, to yield an initial estimate of the motion vector, $\boldsymbol{\mu}_1$. Subsequently, the residuals are adjusted in accordance with this first estimate of the motion vector. The measurements are then re-weighted by applying a bisquare function to the residuals from the previous iteration, giving lower weights to points that do not fit well. Algorithm 1 details the IRLS process. We iterate the reweighted fitting a fixed number n of times until the residuals change only marginally (for the experiments in this paper, $n = 5$), arriving at the latest estimate for the motion vector, $\boldsymbol{\mu}_n$. This can be used to calculate an *intermediate* estimate for the pose update matrix, $\Delta\tilde{\mathbf{E}}$.

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. This is mainly due to the many weighted least squares fitting steps executed, the only component of our algorithm that runs on the CPU.

Still, it is only due to the fact that all of the image processing and edge search takes place on the GPU that we even have the CPU resources to execute the model fitting process several iterations for each frame (3 times for the experiments in this paper to remain at 30fps for 640 $\times$ 480 video). Leveraging the GPU has thus effectively enabled much of the robustness our approach exhibits.

IV. 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 [31] 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. Each fragment program is executed in a single off-screen rendering pass during which the input data is processed. We make use of OpenGL's recent framebuffer extension to efficiently perform render-to-texture and store the resulting output data in GPU memory. Note that merely two CPU-GPU data transfers are performed: one to upload the latest camera image to the GPU initially and one to read back the results of the edge search process, with all processing in between taking place on the GPU. CPU-GPU memory transfers are costly, tend form the main bottleneck in general purpose GPU applications and should generally be avoided.

A. Image acquisition, undistortion & Canny edge detection

We use a firewire camera, standalone or mounted on the robot head, to gather video at a resolution of 640 $\times$ 480 pixels and 30 frames per second. The camera supplies video in YUV 4:1:1 format, which needs to be converted to RGB before image processing begins. Performing color-space conversion on live video on the CPU is a fairly costly operation, yet very parallelizable and thus amenable to GPU implementation. We bind the raw YUV byte array supplied by the camera to a rectangular OpenGL texture which serves as input to a fragment program converting the YUV 4:1:1 image to regular RGB. This is accomplished by selecting the appropriate luminance and chrominance components for each pixel from the input texture and performing a standard color conversion step.

Subsequently, the resulting RGB image is adjusted to account for radial camera distortion. This step is crucial for accurate model-edge to image-edge fitting. Radial distortion can either be taken into account during the fitting process itself or, as we have done, can be dealt with during an image pre-processing step. To perform undistortion, we execute another fragment program taking the RGB texture as input, together with the camera's intrinsic parameters f_x, f_y, o_x, o_y (focal lengths and optical center) and the four radial distortion coefficients $\kappa_1, \ldots, \kappa_4$ and producing an undistorted image as output. Again, since undistortion operates on a per-pixel basis, it is very efficiently implemented on the GPU.

The final step of GPU-based image processing is Canny edge detection. Each of the standard components of the

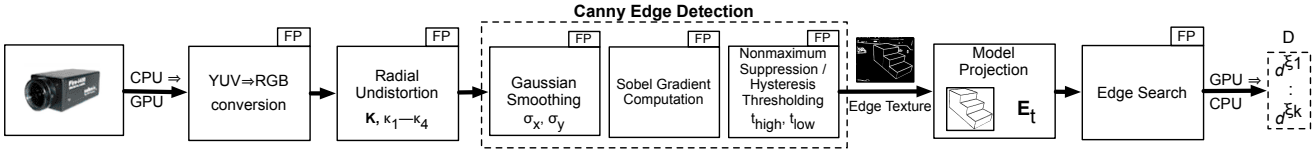


Fig. 2. GPU fragment program cascade defining flow of image processing, model projection and edge search.

edge detector (gaussian smoothing, gradient computation, non-maximum suppression, hysteresis thresholding) executes sequentially as a fragment program. Although implemented on the GPU, the edge detector remains fully parameterizable, with Canny parameters such as $\sigma_{gauss}, thresh_{high}$ and $thresh_{low}$ staying adjustable during tracker operation. The result of GPU-based Canny edge detection is a single binary texture indicating presence or absence of an edge at each pixel.

B. 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 of interest in terms its main salient lines and its faces. Such models are easily generated using CAD or image-based modelling software. We then render our model onto the image plane using OpenGL according to the latest pose estimate. The pose estimate matrix $\mathbf{E}_t$ and the matrix of intrinsic camera parameters $\mathbf{K}$ are converted beforehand to suitable OpenGL modelview and projection matrices, respectively. During rendering, we perform hidden line removal efficiently using depth-buffered OpenGL, resulting in a binary texture containing only the visible edges of the model. The visibility of each control node can then be determined at the entry point to our edge search fragment program by checking for presence of an edge at each control node's (x, y) position in the visible edge texture.

We initialize a certain fixed number of control nodes along the model edges (about 20 per edge for the experiments in this paper, spaced evenly in world coordinates). Control node information is provided to the edge search fragment program as a single four channel RGBA texture. An alternative we are investigating is to use point rendering to reduce the amount of data transferred and the number of times the edge search fragment program is invoked. If a control node is present at a particular (x, y) location in the image and edge-search should be performed at that node, we set the red component of the corresponding pixel in the texture to 255. The direction of the model edge normal at the control node, along which edge search should be performed, is encoded in the remaining components. The green channel indicates the x component of the normal, the blue channel the y component, with the normal scaled such that both components lie in the range $(0, \ldots, 255)$. Since textures cannot contain signed values (the use of full 32-bit floating point textures results in a large performance penalty), we use an integer encoding to represent the signs of the x and y components by a single byte value stored in the alpha channel in the control node texture. The representation of control node data as a texture is illustrated in Figure 3.

Finally, the edge search fragment program steps along the model edge normal trying to detect the closest image edge to the control node in either the positive or negative normal

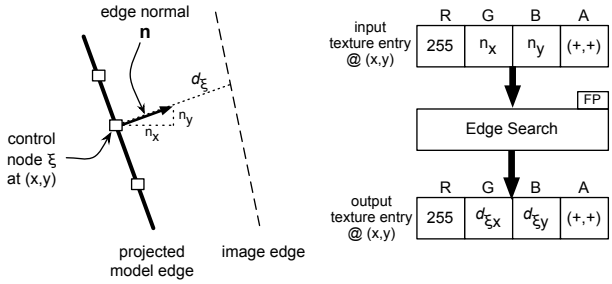


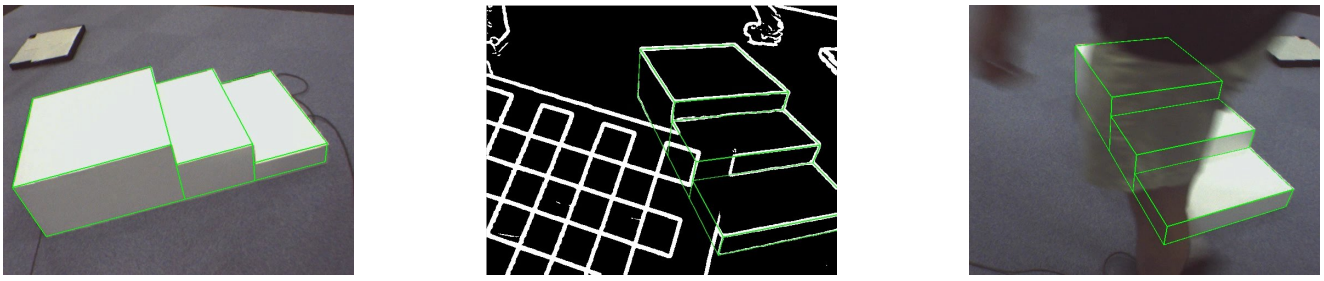
Fig. 3. Representation of input/output data to and from the edge search fragment program.

direction. Search is performed up to a certain cutoff distance. If an image edge is not found within that distance, the control node is ignored and does not contribute to the least squares fitting. If search is successful, the results are stored in an output texture a manner similar to the input data format described above, and shown in Figure 3. The red channel indicates success of search, the green and blue channels hold the vector leading from the control node to the closest edge, and the alpha channel is used for sign encoding. While some prior approaches have, for reasons of computational tractability, restricted the edge search to the 45° line closest to the true edge normal, our approach does indeed consider the true model edge normals, albeit quantized to pixel coordinates, resulting in increased accuracy.

It is worth noting that both the search distance and the number of control points search is performed on directly affect the running time of the edge search process. Previous approaches have thus tended to keep these numbers small (e.g. 20 pixels distance cutoff and one control node every 20 pixels along the model edge). While it is certainly not wise to increase these numbers arbitrarily (as control points will start latching onto distant background edges), even with many hundreds of control points and edge search distances of up to 50 pixels, we have not detected any slow-down in our GPU implementation.

V. Robot Localization & Environment Mapping

To perform navigation planning for our robot we need a representation of the environment, with the robot localized within that representation. Suppose that we are interested in objects that a humanoid should avoid during walking or sets of stairs that it should climb during locomotion. We can establish a map coordinate system in which the object being tracked is assumed to remain at a fixed position and orientation in map coordinates, 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



(a) (b) (c)

Fig. 4. 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)*.

positioning the camera in a consistent coordinate system relative to the object of interest.

For footstep planning, knowing the accurate location of the robot foot at any point during execution is crucial. Fortunately, the robot kinematics can supply us with another transform, ${}^{c}_{f}T$, locating the robot foot relative to the camera at any instant in time. This transform, when chained with ${}^{m}_{c}T$, allows us to recover the position and orientation of the foot in map coordinates.

Given the fixed position and orientation of the tracked objects in map coordinates, it is straightforward to generate a height map describing the robot environment. We render our object model using an orthographic projection and depth buffering at the appropriate pose ${}^{m}_{o}T$ and viewed from an OpenGL camera positioned above and looking straight down onto the scene. A height map can then be generated simply by reading back the depth buffer after rendering and scaling the resultant depths using the values for $zNear$ and $zFar$ used for the orthographic projection. Due to its object-centric nature, the height map need only be constructed once. Together with the continuously updated robot foot location, this map is used to find a safe path through the environment.

VI. Heightmap-based Footstep Planning

The navigation planning performed for the experiments in this paper uses a modified version of our previously described footstep planner [32]. The planner reduces the problem of motion planning for the humanoid to planning a sequence of footsteps for the robot to follow. Given a sequence of step locations, a walking controller then generates a dynamically stable motion to walk along the desired path.

The planner takes as input a 2.5D height map representation of the environment, a start and goal state, and a model of the the robot's capabilities. It returns a path of footholds in the world which take the robot from its current state to the specified goal. For planning purposes, we model the humanoid as having a set of actions available to it for each footstep. These actions represent locations – (x, y, θ) relative to the stance foot – where the robot can place its foot for the next step. Each action has several associated attributes, such as the height that it can step up or down, the height of obstacles it can traverse, and the cost of taking that action. During planning, each candidate footstep is evaluated to determine if it is both a stable place to step, and if the intervening terrain allows that action to be taken. To account for environments for which the given actions may not "fit," a local search is performed at each step location to find the closest location to which the robot can safely step. This search allows the planner to adjust its actions during planning to walk up a set of stairs, without the need for an action which matches the length of the stairs.

Once the footstep path has been found, we calculate the convex hull of the terrain between successive locations for each foot. We use this convex hull to generate knot points for a cubic spline swing leg trajectory which will move the foot smoothly from one foothold to the next while avoiding any obstacles between them. In addition, timings are calculated for each step so as to slow the walking down when making long steps or when stepping up or down. These adjustments keep the robot within the joint velocities and allow it to execute the desired trajectory.

The path, together with the swing leg trajectories and step timings, is then used by the walking and balance controller to generate a dynamically stable walking motion which can safely take the robot from its current state to the goal location in the environment.

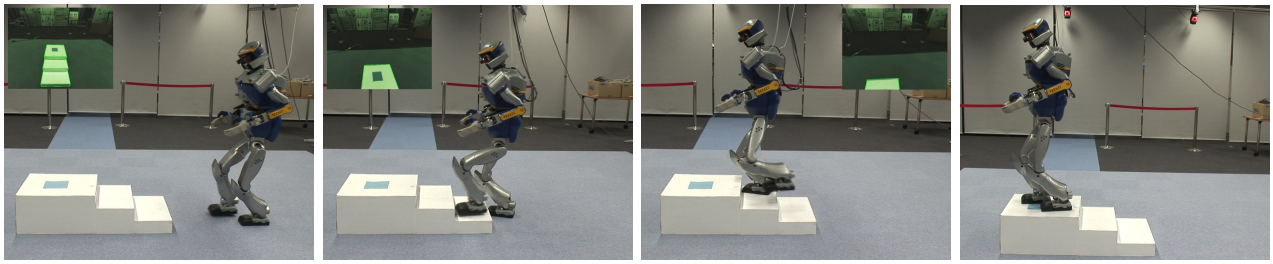
VII. Results

A. Standalone tracker operation

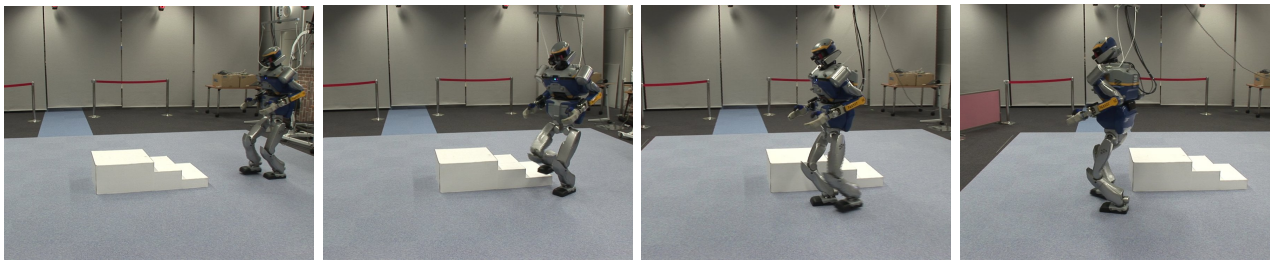
To establish the operational performance of our tracker, we first used a standalone firewire camera attached to a commodity PC equipped with an NVIDIA GeForce 7800GTX PCI-Express GPU (although our system has been tested to work on a range of modern GPUs from various manufacturers). The system tracked a set of white stairs at 30fps while an experimenter moved the handheld camera around freely. Before starting the tracking process, we initialize the pose of the stair model to roughly reflect the actual pose of the stairs in the camera view. This is done via a GUI and need not be very accurate. Upon tracker initialization, the model edges almost instantaneously "snap" to the image edges, subsequently tracking the stairs and recovering their pose. 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 4(a) shows a typical view with the tracked model superimposed in green, Figure 4(b) shows a view of the model superimposed on the extracted image edges during object occlusion with a checkerboard. Figure 4(c) shows a view of the tracker during severe occlusion by an experimenter walking in front of the camera.

B. 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.



(a)



(b)

Fig. 5. 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)*.

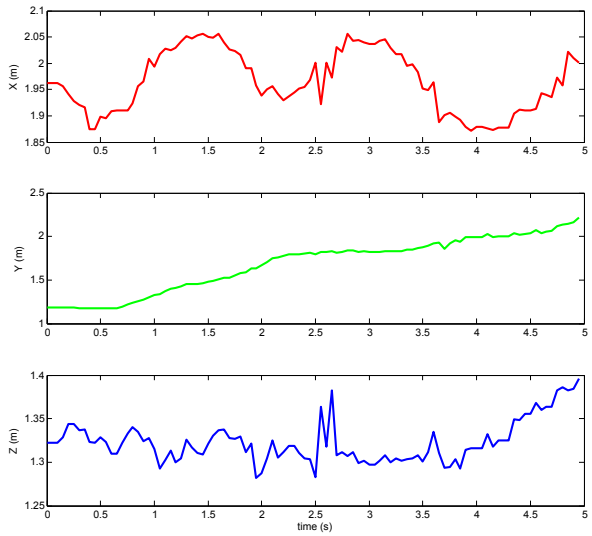


Fig. 6. Plots of the x, y and z coordinates of the camera in map coordinates during a walking sequence of HRP-2 approaching and starting to climb a set of stairs.

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. After the tracker builds an environment map and localizes the robot, the footstep planner proceeds to compute an optimal sequence of footsteps toward the goal positioned at the top of the stairs (for stair climbing) or on the opposite side of the stairs (for obstacle avoidance), which is then executed by the humanoid. Since the object of interest is tracked continuously throughout the walking sequence (provided it remains in view), the robot's location can be updated in real-time during the walking sequence.

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 5(a) shows HRP-2 successfully approaching and climbing our set of stairs. Figure 5(b) shows HRP-2 navigating around the same set of stairs. Note that, unlike in several previous approaches to stair climbing, neither planner nor controller have any predefined knowledge about the stair geometry or location, but instead operate only on the map and localization data supplied by the tracker. Also note that our tracker continues to recover the stair pose despite significant camera motion occurring during the walking sequence and even when the stairs have nearly vanished from the camera view.

Figure 6 plots the tracker-reconstructed (x, y, z) camera position in map coordinates during the initial part of a straight-on stair climbing sequence, until the stairs fall outside of the robot's view. Note the increasing y coordinate as the robot approaches the stairs directly in front of it in map coordinates and the increasing z coordinate toward the end when the robot begins to climb the stairs. The oscillations in x reflect the lateral swaying of the robot as it changes stance foot during walking. The overall lack of smoothness in the curves hints at the shakiness inherent in perception on a rigid, moving humanoid platform. Our tracker handles both of these issues well. There are of course limits to the amount of motion that can occur between two frames before the system loses track of the object. The spike in x and z at around $2.5s$ shows an instance where the object pose was briefly distorted (but later recovered) due to a particularly harsh head motion resulting from a fast sidestepping movement executed by the robot.

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

As an improvement, we would like to eliminate the currently necessary manual pose initialization step. Since a rough model-to-image matching is usually enough to start the tracker, simple appeareance-based initialization (e.g. by matching an image taken from a database of image/pose pairs to the camera view) may be enough. Secondly, for humanoids to navigate truly cluttered environments, we would like to track many scene objects concurrently. Current tracker performance seems to indicate that there are still plenty of GPU resources available to deal with more objects, but that increasing the number of GPU-CPU readback steps required during IRLS fitting to deal with multiple objects would be prohibitive. We are thus investigating the possibility of performing robust linear regression entirely on the GPU. Third, we are working towards tighter interplay between our 3D tracker and a footstep planner supporting efficient replanning to deal with dynamic environments where tracked objects move quickly and unpredictably during robot navigation.

Finally, we would like to exploit our tracker for other humanoid tasks such as visual servoing for grasping or perhaps even person tracking for human-robot interaction applications. We believe that GPUs will play an increasingly important role as an implementation platform for robotic perception algorithms, enabling humanoid robots to perform increasingly complex tasks in everyday, real-world environments.

IX. Acknowledgements

This work was supported by an NVIDIA Fellowship on the part of the first author, as well as the Robotics Institute of Carnegie Mellon University and the Digital Human Research Center. This material is based upon work supported in part by the National Science Foundation under grant EEC-0540865.

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, "Fitting parameterized three-dimensional models to images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-13, no. 5, pp. 441–450, 1991.

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

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

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

[7] 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.

[8] ——, "Real-time visual tracking of complex structures," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 932–946, 2002.

[9] V. Kyrki and D. Kragic, "Integration of model-based and model-free cues for visual object tracking in 3d," *Proc. of the IEEE Int. Conf on Robotics and Automation (ICRA'05)*, pp. 1554–1560, April 2005.

[10] 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.

[11] 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.

[12] 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.

[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] M. Yagi and V. J. Lumelsky, "Biped robot locomotion in scenes with unknown obstacles," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'99)*, Detroit, MI, May 1999, pp. 375–380.

[15] 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.

[16] 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.

[17] 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.

[18] J.-S. Gutmann, M. Fukuchi, and M. Fujita, "Stair climbing for humanoid robots using stereo vision," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'04)*, Sendai, Japan, September 2004, pp. 1407–1413.

[19] O. Lorch, J. Denk, J. F. Seara, M. Buss, F. Freyberger, and G. Schmidt, "ViGWaM - an emulation environment for a vision guided virtual walking machine," in *Proc. of the IEEE-RAS/RSJ Int. Conf. on Humanoid Robotics (Humanoids 2000)*, 2000.

[20] R. Cupec, O. Lorch, and G. Schmidt, "Vision-guided humanoid walking - concepts and experiments," in *Proc. of the 12th Int. Workshop on Robotics in Alpe-Adria-Danube Region (RAAD'03)*, Cassino, Italy, May 2003.

[21] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, "The development of Honda humanoid robot," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'98)*, May 1998, pp. 1321–1326.

[22] K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue, "Toe joints that enhance bipedal and fullbody motion of humanoid robots," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'02)*, Washington, DC, USA, May 2002, pp. 3105–3110.

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

[24] 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.

[25] 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.

[26] 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.

[27] Z. Zhang, "Flexible camera calibration by viewing a plane from unknown orientations," in *Proc. of the Int. Conf. on Computer Vision (ICCV '99)*, Corfu, Greece, September 1999, pp. 666–673.

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

[29] V. Varadarajan, *Lie Groups, Lie Algebras and Their Representations*, ser. Graduate Texts in Mathematics. Springer-Verlag, 1974, no. 102.

[30] P. Preisig and D. Kragic, "Robust statistics for 3d object tracking," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'06)*, Orlando, FL, USA, May 2006, pp. 2403–2408.

[31] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard, "Cg: a system for programming graphics hardware in a c-like language," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 896–907, 2003.

[32] 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.