It just goes to show you, there's always something.
If it's not one thing, it's another.

— Gilda Radner as Roseanne Rosanna-Danna
from Saturday Night Live (1977)

Appendix A

Angle Between Rays Computation from Step 7 of the Stability Margin Algorithm

The following is a more detailed version of Step 7 of the Stability Margin Algorithm presented in Section 4.4:

Calculate the included angle between each ray and its successive ray (the ray through the next sequential convex hull vertex), all the way around the convex hull.

Note that in Step 6 of the algorithm, rays were defined from the **X-Y** projection of the robot's center of gravity point to each of the convex hull vertex points. In the code these rays are actually vectors defined as:

$\mathbf{V}_k = \mathbf{P}_k - \mathbf{P}_{CG}$ where k is the number of convex hull vertices

FOR each sequential pair of vectors around the polygon (including the angle between the first vector and the last vector):

a) Compute angle between successive vectors.

$$angle = \cos^{-1}\left(\frac{\mathbf{V}_{k-1} \bullet \mathbf{V}_k}{|\mathbf{V}_{k-1}||\mathbf{V}_k|} \right)$$

b) $\mathbf{C} = \mathbf{V}_{k-1} \times \mathbf{V}_k$

c) **IF** $c_z < 0$

THEN *angle* is NEGATIVE.

ELSE

angle is POSITIVE.

END of Step 7.

Note that in order to easily handle the case of the angle between the last vector and the first vector, it is useful to store the convex hull vertices as a circular linked list or as a linear array with an extra member added to the end that is a duplicate of the first vertex.

Appendix B

Distance Computation from Step 10 of the Stability Margin Algorithm

The following is a more detailed version of Step 10 of the Stability Margin Algorithm presented in Section 4.4:

Compute the minimum **X-Y** distance, d_k, from the center of gravity, $\mathbf{P}_{CG}$, to each line, $\overline{\mathbf{P}_k\mathbf{P}_{k+1}}$, that connects sequential vertices on the convex hull.

Where d_k is computed as follows:

a) $e = p_{x_{k+1}} - p_{x_k}$

b) **IF** $e = 0$

THEN $d_k = p_{x_{CG}} - p_{x_k}$

This operation handles the situation of a line segment parallel to the Y-axis, and prevents divide by zero errors below.

ELSE

c) $$a = \frac{\left(p_{y_k} - p_{y_{k+1}}\right)}{e}$$

d) $c = -a\,p_{x_k} - p_{y_k}$

e) $$d_k = \frac{\left| a\, p_{x_{CG}} + p_{y_{CG}} + c \right|}{\sqrt{a^2 + 1}}$$

END of Step 10.

Appendix C

Defining Leg Pair Movements Relative to the Leg Pair's Current Position

This appendix explains an alternative technique for defining new positions of the robot leg pairs to the one presented in Chapter 4. While this method is less efficient and appears to be ill-suited for use with automated leg step trajectory generation, it is included here, because an earlier version of the simulation code used this method, and, more importantly, because this method may be useful for teleoperating the robot and for performing guarded motions for adapting to the terrain.

Recall that the problem is: given the position of leg pair l at time t, define where it will be at the next time increment, $t + \Delta t$. The method presented in Section 4.5.1 accomplishes this by making a new homogeneous transform with respect to the *local* coordinate frame. However, with this alternative method, movement definition is done by making a new transform, ${}^{l'}_{l}\mathbf{T}$, which describes the new position of the lth leg pair relative to its *current* position. This method may be advantageous in cases in which a teleoperator is trying to specify the motion of the vehicle based on images transmitted from cameras mounted directly on the vehicle.

Figure C.1 shows an example of specifying a new position for the robot's second leg pair frame (position 2'). The difference between this method and the one presented in Ch. 4 can be seen graphically by comparing Fig. C.1 with Fig. 4.9.

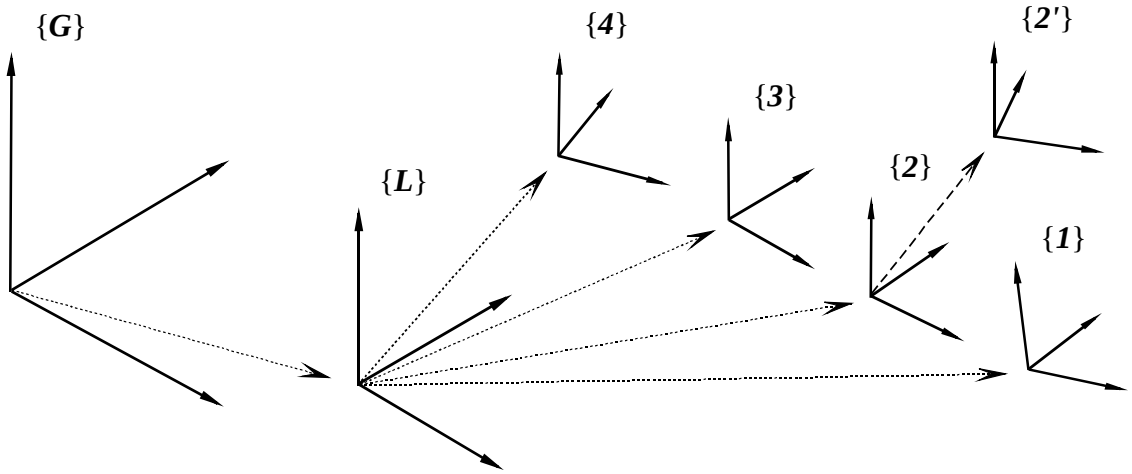

Figure C.1 - Specifying a New Position for Leg Pair 2 Relative to its Current Position

The ${}^{l}_{l'}\mathbf{T}$ transformation is constructed using the same basic mathematics that was used to make the ${}^{L}_{l'}\mathbf{T}$ transform explained in Section 4.5.2. The key difference is that the motion program interpreter must be modified to output the 6 DOF movement specification (x, y, z, γ, β, α) with respect to the *current* leg pair coordinate frame, rather than the local frame.

By modifying Eq. 4.8 to indicate these new input variables, we obtain the matrix defining the transformation from l' to l:

$$ {}^{l}_{l'}\mathbf{T} = \left[\begin{array}{ccc:c} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma & {}^{l}p_x \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma & {}^{l}p_y \\ -s\beta & c\beta s\gamma & c\beta c\gamma & {}^{l}p_z \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] \qquad \text{(C.1)} $$

Since the new frame of reference, l', replaces the current frame, l, the transformation from the leg pair frame to the local frame must be updated to reflect the new position of the leg pair. This is accomplished using the equation below.

$$ {}^{L}_{l'}\mathbf{T} = {}^{L}_{l}\mathbf{T}\ {}^{l}_{l'}\mathbf{T} \qquad \text{for } 1 \le l \le N \tag{C.2} $$

Where N is the number of leg pairs on the robot. Note that all leg pairs that move during a particular time interval must be updated. This result can then be directly used in the inverse kinematics, stability analysis, and rendering computations mentioned in Ch. 4.

Thus, at each time increment, this method requires an additional matrix multiplication for every moving leg pair when compared to the method described in Ch. 4. These matrix multiplications can be simplified some because the last row of the homogeneous matrices is constant, even when multiplied. So instead of 64 multiplications and 48 additions for the general multiplication of two [4x4] matrices, this operation requires 39 multiplications and 27 additions.

Handling single discrete movements using this method is simple enough, so long as the matrix multiplications can be done quickly enough to maintain a reasonable update rate. If the update rate gets too slow, the control lags will cause the teleoperator to “oversteer”. However, using this method for tracking a pre-planned leg step trajectory relative to the local or global frame is cumbersome. This difficulty results because the “current” frame of reference changes at *every* point along the trajectory. To track pre-planned trajectories using this method involves several more computational steps than have been presented here. The preferred technique for specifying pre-planned trajectories was presented in Section 4.5.1.