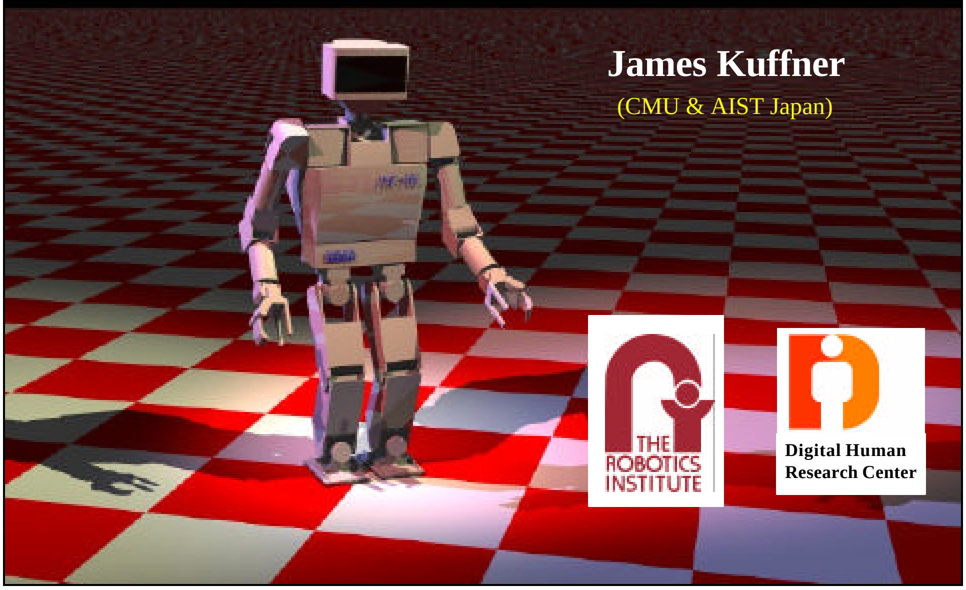
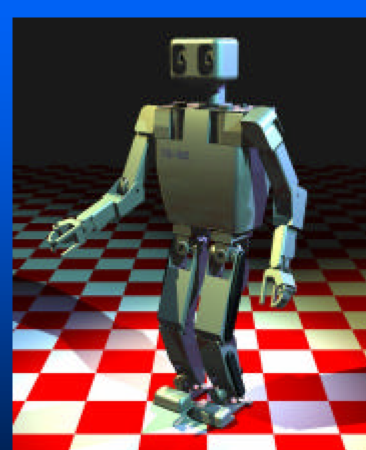
# Self-Collision Detection and Motion Planning for Humanoid Robots




**James Kuffner**
(CMU & AIST Japan)

## Talk Overview

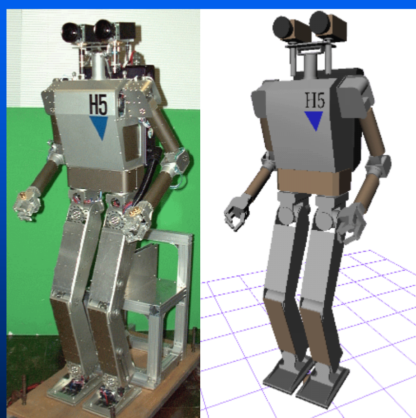
- Self-Collision Detection
- Feature-based Minimum Distance Computation:
  - Approximate Geometric Models
  - Minimum Distances for 3D Convex Polyhedral Models
  - Coherency and Efficiency
- Safe Biped Walking
- Motion Planning Applications
- Conclusion



# JSK Humanoid Research Platforms

*1998 - 1999*

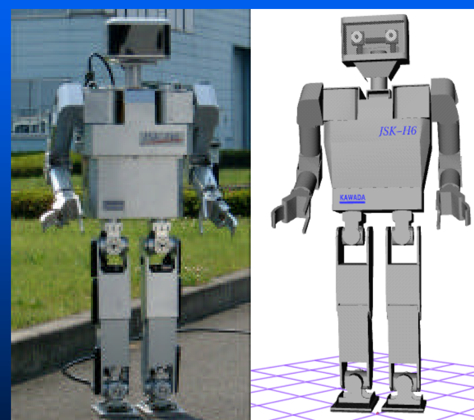
H5 (30 DOF ; 33kg ; 1.27m )



[ Nagasaka, Konno, Nishiwaki, Kitagawa, Sugihara, Kagami, Inaba, Inoue ]

*2000 - 2001*

H6 (35 DOF ; 55kg ; 1.37m )



[ Nishiwaki, Sugihara, Kagami, Kanehiro, Inaba, Inoue ]

# Humanoid "H7"

*2001 - 2003*

# Online Walking Trajectory Generation

- Even without obstacles, balance criteria alone cannot guarantee safe motion!
- Need methods for detecting and preventing leg interference and body self-collision
  - Efficient (low computation time)
  - Reliable (computation time has low variance)
  - Numerically robust
- Should be conservative (error-bounds)

# Self-Collision

- Occurs when one of more of the links of a robot collide
- Dangers of self-collision:
  - Can cause robot to damage itself
  - Through a loss of balance or control, can cause damage to the surrounding environment (including nearby people)
- *Detecting* and *preventing* potential self-collisions is fundamental to developing safe robots

# Safe Biped Walking

IDEAL:

– A colliding posture is *never output* by the controller, thus all trajectories executed by the robot are *guaranteed* to be free of self-collision

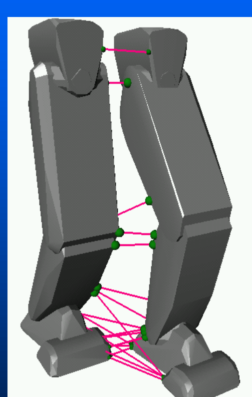
CURRENT:

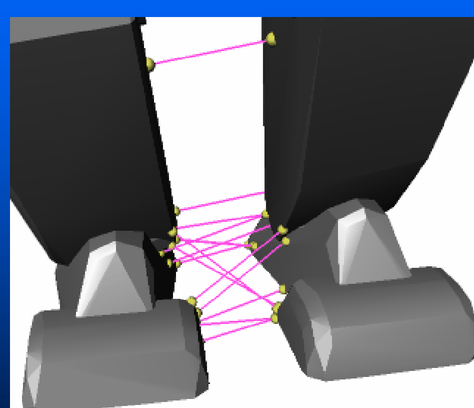
– All computed *desired* body trajectories are free of self-collision (prior to online balance compensation and PD control)

– Depends on the accuracy of both the kinematic and dynamic model of the robot and environment

# A Minimum Distance Approach

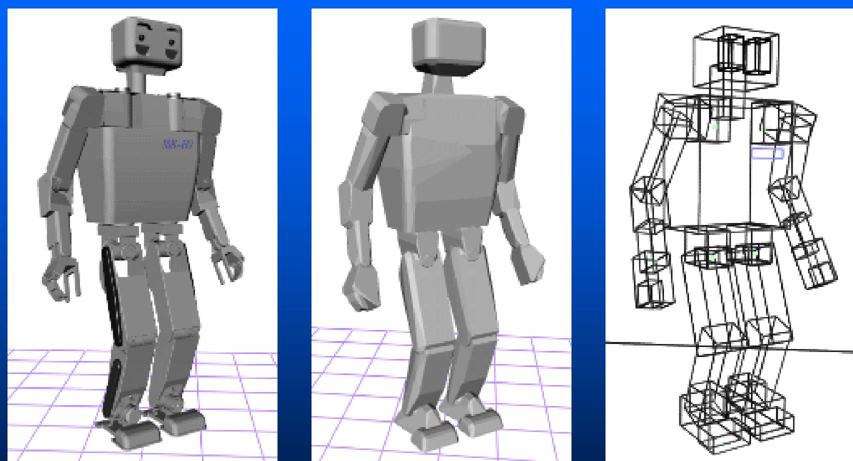
Maintain closest points between pairs of links





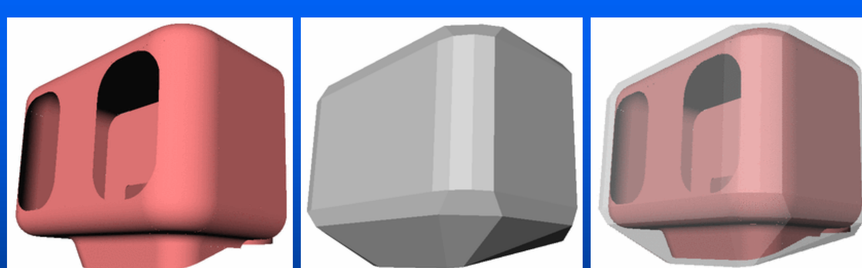
All pairs can be considered, or a subset of possibly-colliding pairs can be pre-computed **[Kanehiro, Hirukawa '01]**

## Approximating Link Geometry



| Full CAD Model | Convex Hulls | Bounding Boxes |
|---|---|---|
| ( 314,588 triangles ) | ( 2,702 triangles ) | ( 432 triangles ) |

## Protective Convex Hulls



| CAD model | Protective Hull | Combined View |
|---|---|---|

# Self-Collision Detection

- **GIVEN:**
  – Tree-like kinematic structure with N links:
- **ASSUME:**
  – joint limits specified to prevent collisions between adjacent links.
- Remaining pairs to check:

$$\frac{N(N-1)}{2} - (N-1) = \frac{N^2 - 3N + 2}{2}$$

30 links = 454 pairs

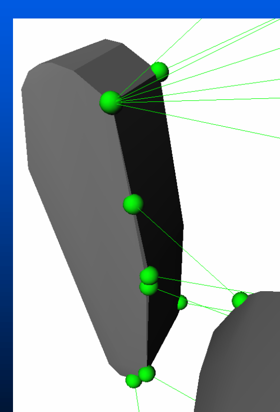
# Minimum Distance Computation

**CONVEX POLYHEDRA:**

- GJK [Gilbert, Johnson, Keerthi '88]
- Lin-Canny [Lin, Canny '91]
- Enhanced-GJK [Cameron '97]
- V-Clip [Mirtich '98]
- H-walk [Guibas, Hsu, Zhang '99]

**NON-CONVEX POLYHEDRA:**

- Sphere Trees [Quinlan '94]
- CFL [Kawachi, Suzuki '00]
- SWIFT++ : [Ehmann, Lin '01]



# Feature-based Minimum Distance Computation

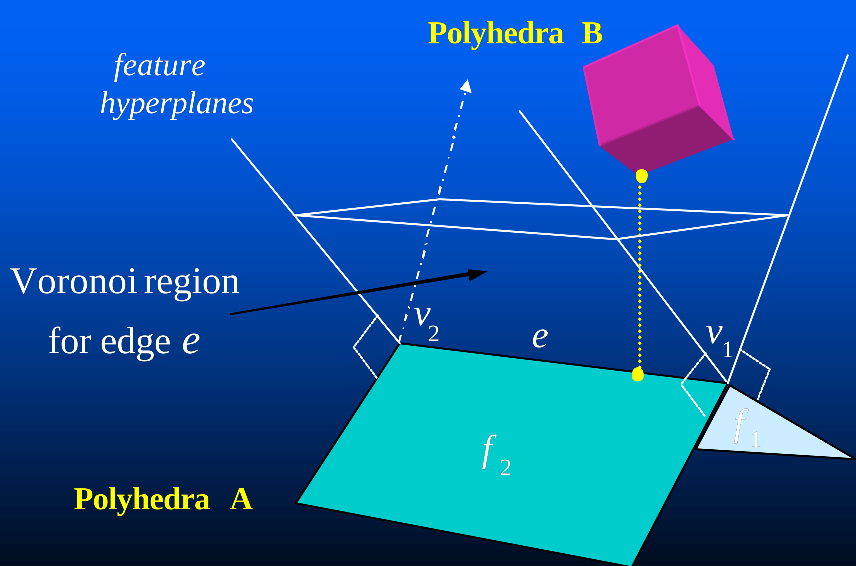
**ADVANTAGES:**

- Exploits spatial and temporal coherency
- Minimum distance value and pairs of closest feature points can be updated in "almost constant" time
- No performance loss for near-colliding cases

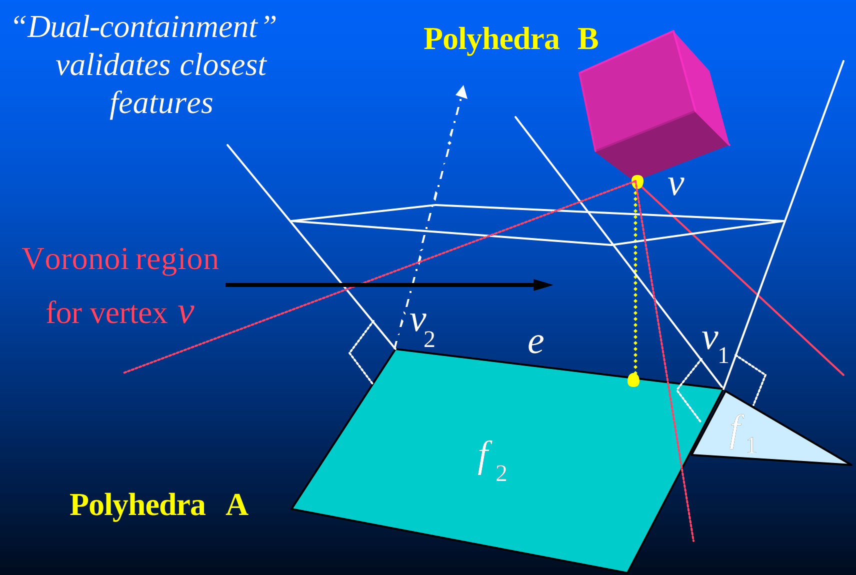
**DISADVANTAGES:**

- Typically limited only to closed, bounded, convex polyhedra (or hierarchical collections of them)

# Voronoi-based Minimum Distances



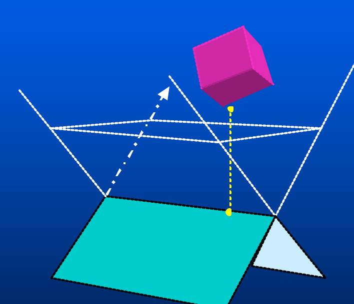
## Voronoi-based Minimum Distances




## Voronoi-based Minimum Distances

POSSIBLE STATES:

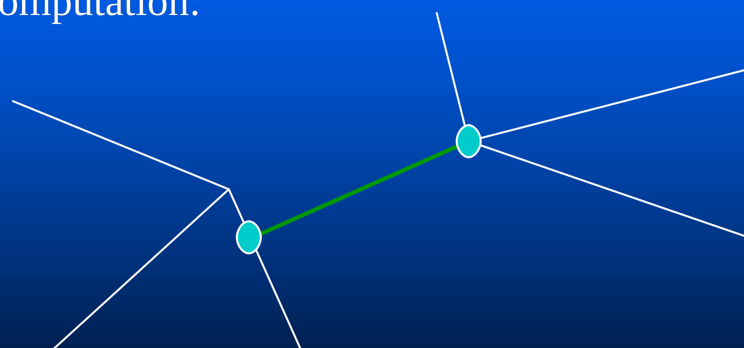
- Vertex – Vertex (V-V)
- Vertex – Edge (V-E)
- Vertex – Face (V-F)
- Edge – Edge (E-E)
- Edge – Face (E-F)



Algorithm keeps track of possible transitions between states

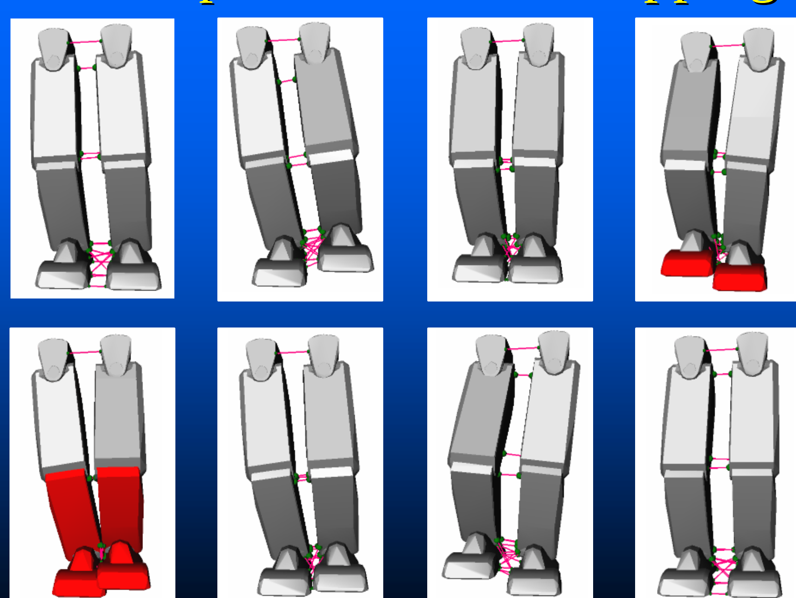
# Coherency and Efficiency

- Closest feature pairs cached from previous computation:



Subsequent checks verify Voronoi containment constraints

# Example 1 : In-Place Stepping

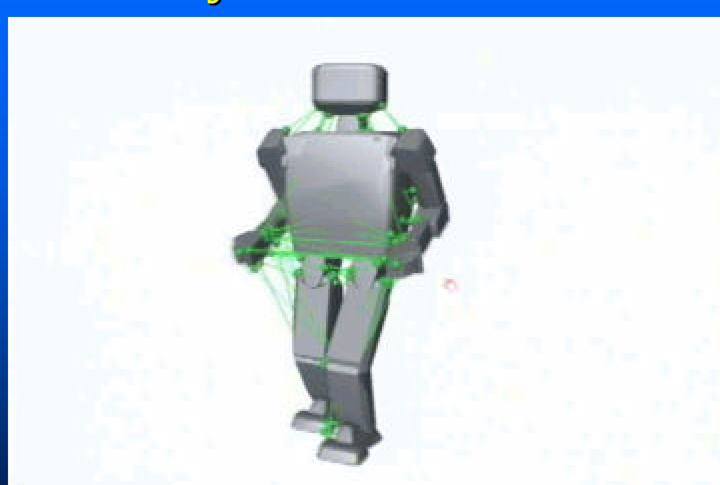


# In-Place & Forward-Turning Step

14 links w/ 14 DOF
Select = 19 active pairs
Full = 73 active pairs

866 MHz dual Pentium III
Time per config = 0.128 ms
Time per config = 0.429 ms

# Whole-Body Self-Collision Detection



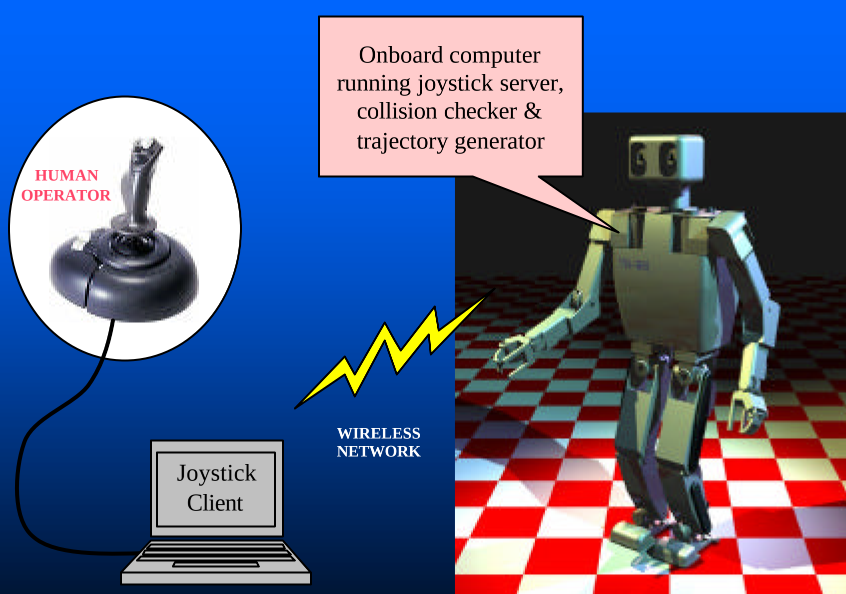
31 links w/ 30 DOF
76 active pairs

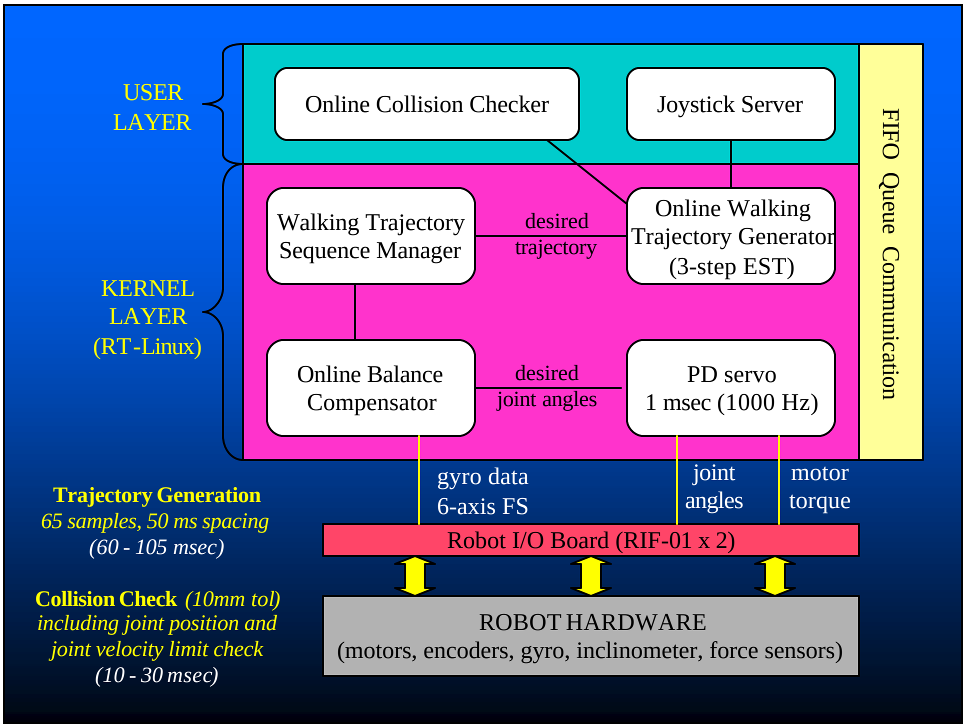
866 MHz dual Pentium III
Time per config = 0.44 ms

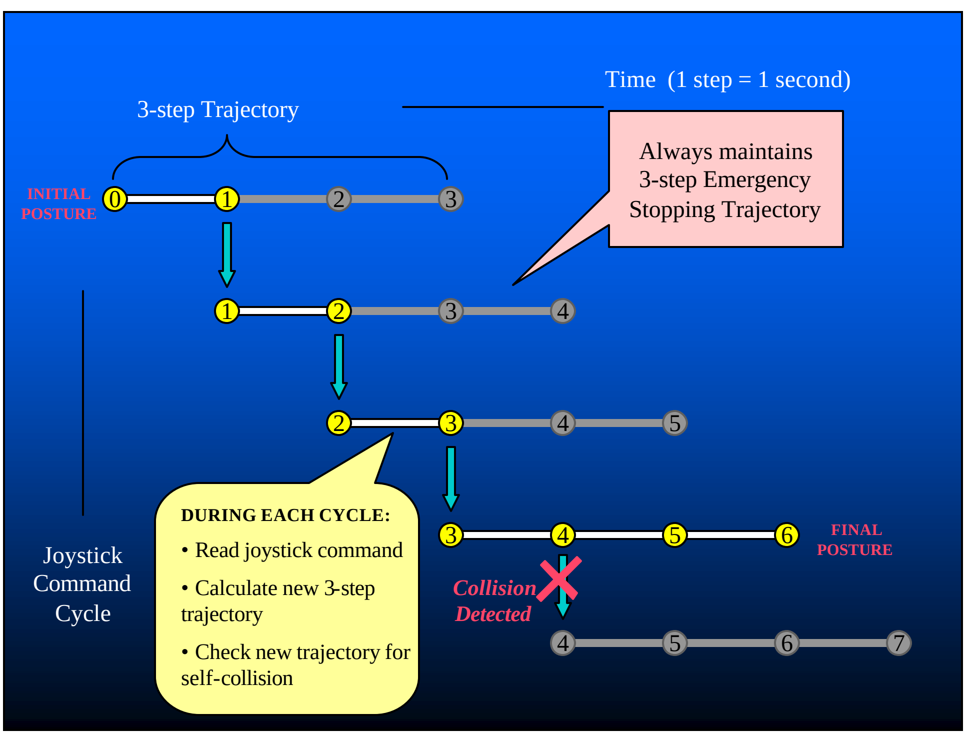
# Whole-Body Self-Collision Detection

31 links w/ 30 DOF
Select = 76 active pairs
Full = 425 active pairs

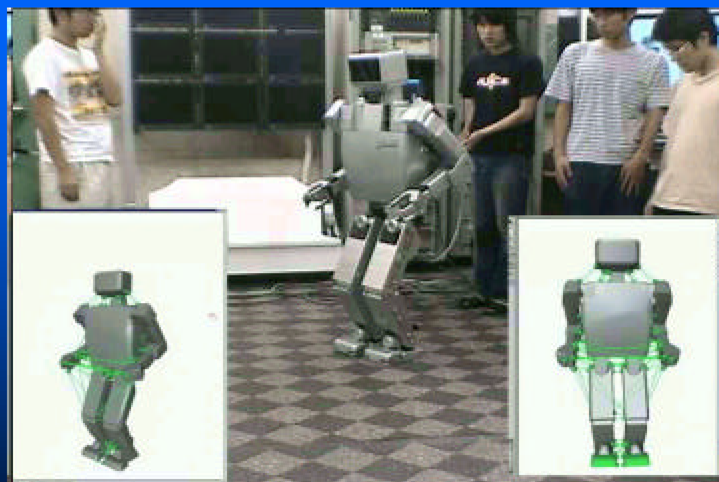
866 MHz dual Pentium III
Time per config = 0.44 ms
Time per config = 2.44 ms




USER LAYER
KERNEL LAYER (RT-Linux)
Online Collision Checker
Joystick Server
FIFO Queue Communication
Walking Trajectory Sequence Manager
desired trajectory
Online Walking Trajectory Generator (3-step EST)
Online Balance Compensator
desired joint angles
PD servo 1 msec (1000 Hz)
gyro data 6-axis FS
joint angles
motor torque
Robot I/O Board (RIF-01 x 2)
ROBOT HARDWARE (motors, encoders, gyro, inclinometer, force sensors)
Trajectory Generation
65 samples, 50 ms spacing
(60 - 105 msec)
Collision Check (10mm tol)
including joint position and
joint velocity limit check
(10 - 30 msec)


Time (1 step = 1 second)
3-step Trajectory
Always maintains 3-step Emergency Stopping Trajectory
INITIAL POSTURE
FINAL POSTURE
Joystick Command Cycle
DURING EACH CYCLE:
• Read joystick command
• Calculate new 3-step trajectory
• Check new trajectory for self-collision
Collision Detected

# Safe Online Joystick Control



Self-collision check
3 steps in the future
(60-65 configurations)

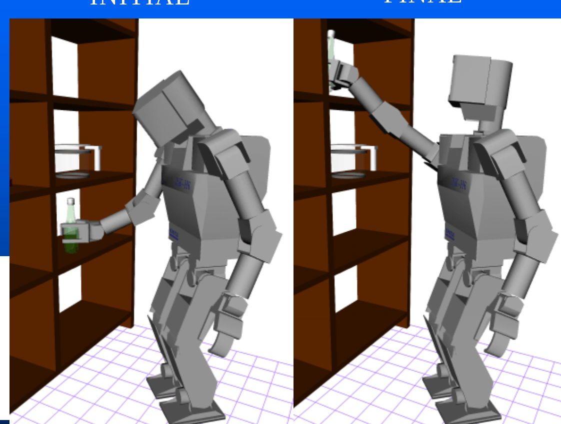
Minimum Distance
Configuration

# Task-Based Control : Manipulation

High-level Goal:
*"Move object* X *from point* A *to point* B"

Software Components:
- Grasp Selection
- Inverse Kinematics
- Path Search

INITIAL FINAL

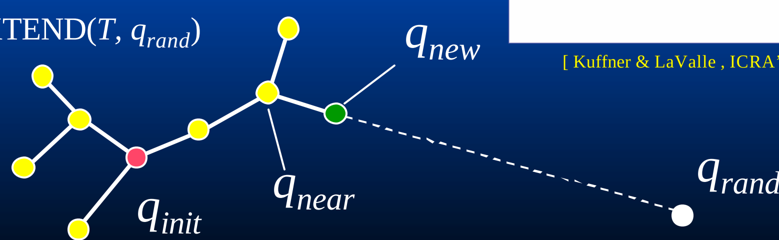


Compute kinematic path by searching arm C-space

# Path Planning with RRTs (Rapidly-Exploring Random Trees)

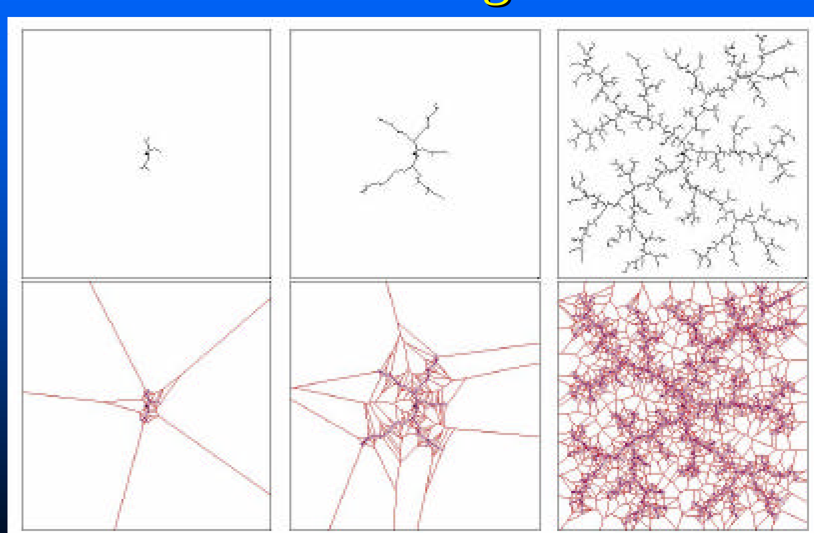
BUILD_RRT ($q_{init}$) {
  $T.init(q_{init})$;
  for $k$ = 1 to K do
    $q_{rand}$ = RANDOM_CONFIG();
    EXTEND($T$, $q_{rand}$)
}

EXTEND($T$, $q_{rand}$)

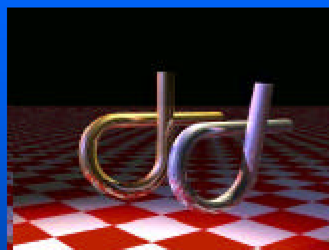


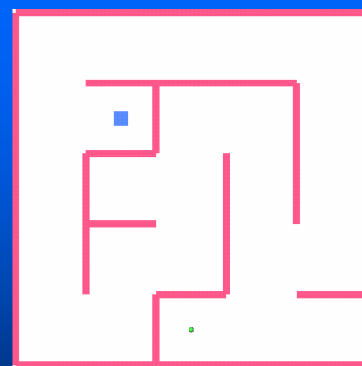

[ Kuffner & LaValle , ICRA'00]

# RRTs Biased to Explore Large Voronoi Regions

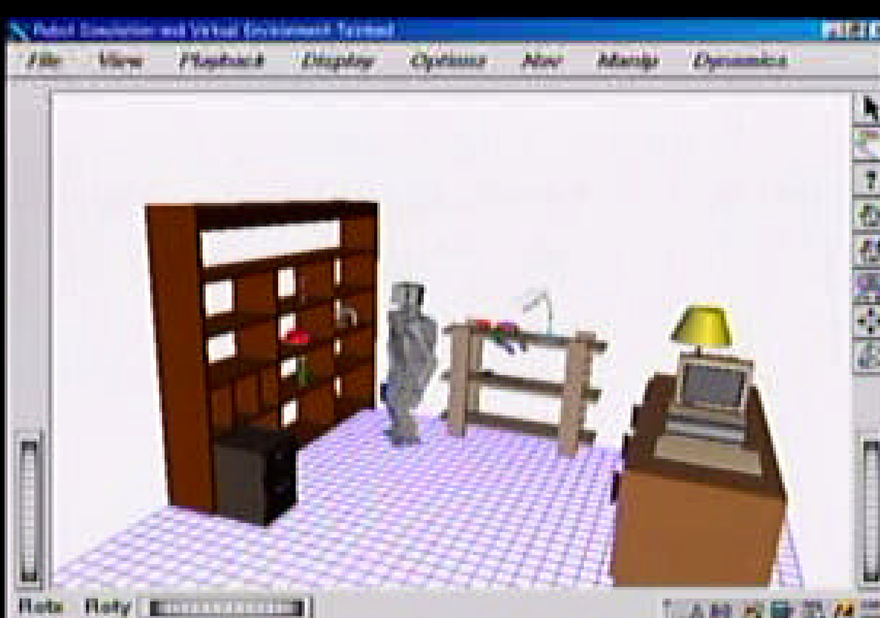


# Grow two RRTs towards each other

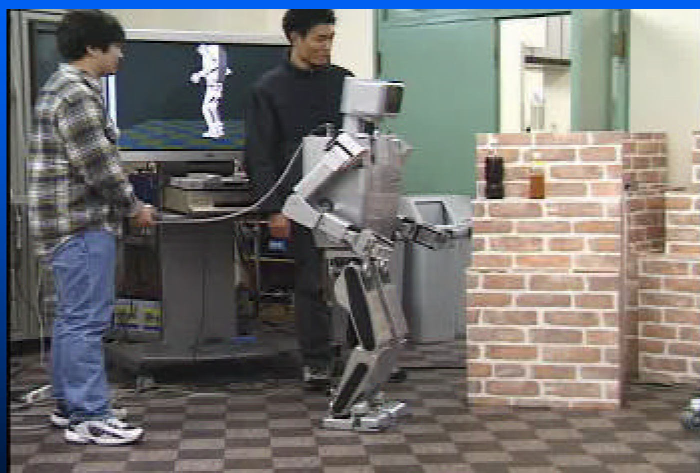




[ Kuffner, LaValle ICRA '00]

# Online Manipulation Planning



# Other Applications : Motion Planning

- High-dim. C-space (30 DOF)
- Balance and obstacle constraints restrict space of feasible configurations

GOAL: Compute a trajectory that:

- Connects the initial and final posture
- Collision-free *(including self-collision)*
- Dynamically-stable

# Dynamic Stability Criteria

- *Static stability:*

  $dC_x(d?) = 0$

  $dC_y(d?) = 0$

$\mathbf{C} = \mathbf{X}(?)$

$\dot{\mathbf{C}} = \mathbf{J}(?)\dot{?}$

$\mathbf{J}(?) = \frac{\partial \mathbf{X}(?)}{\partial ?}$

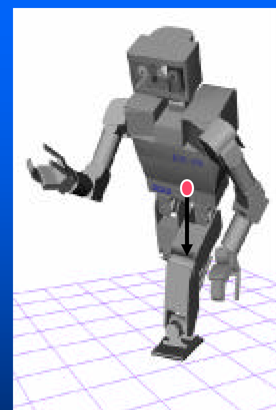
$d\mathbf{C} = \mathbf{J}(?)d?(t)$

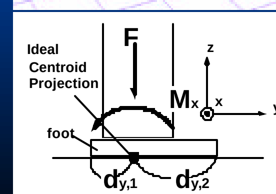
- *Dynamic stability:*

  Moment of Inertia Contstraints

  Zero Moment Point (ZMP)

$$-d_{y,1}F < M_x(d?) < d_{y,2}F$$
$$-d_{x,1}F < M_y(d?) < d_{x,2}F$$
$$-mF < M_z(d?) < mF$$

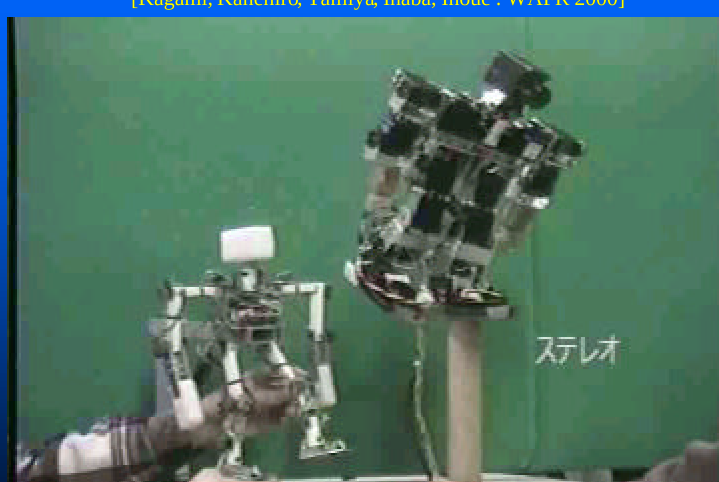





# Balance Compensation as an Optimization Problem

AutoBalancer: online dynamics filter

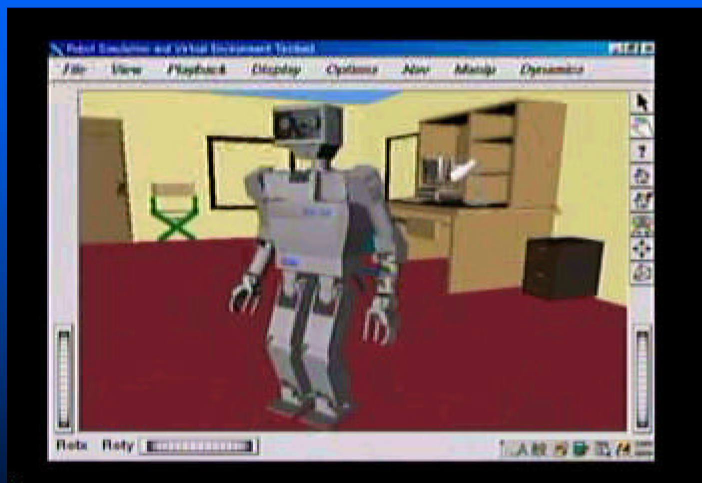
[Kagami, Kanehiro, Tamiya, Inaba, Inoue : WAFR 2000]




$$d? = d?_{cmd} + d?_{ret} + d?_{comp}$$

$$E(d?_{comp}) = \sum_{i=1}^{p} w_i (d?_{comp,i})^2$$

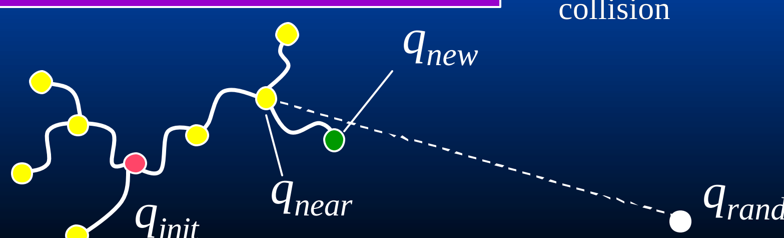
# “Virtual Puppet” Interface



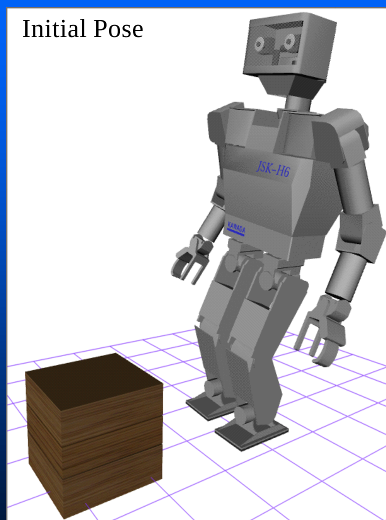
# Modified RRT-Connect

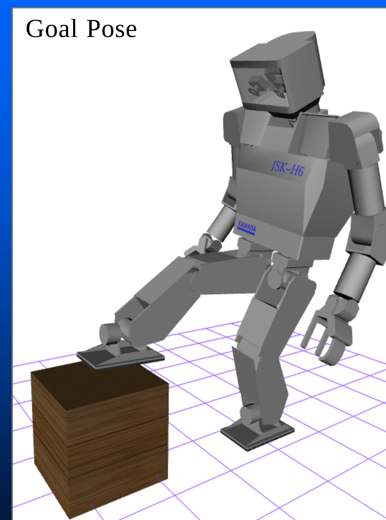
```
RRT_CONNECT_STABLE (q_init, q_goal) {
  T_a.init(q_init);   T_b.init(q_goal);
  for k =  1 to K do
    q_rand = STABLE_CONFIG();
    if not (EXTEND(T_a, q_rand) = Trapped) then
       if (EXTEND(T_b, q_new) = Reached) then
          Return PATH(T_a, T_b);
    SWAP(T_a, T_b);
  Return Failure;
}
```

- Verify obstacle and balance constraints for each incremental step motion
- target configurations are statically-stable postures without self-collision

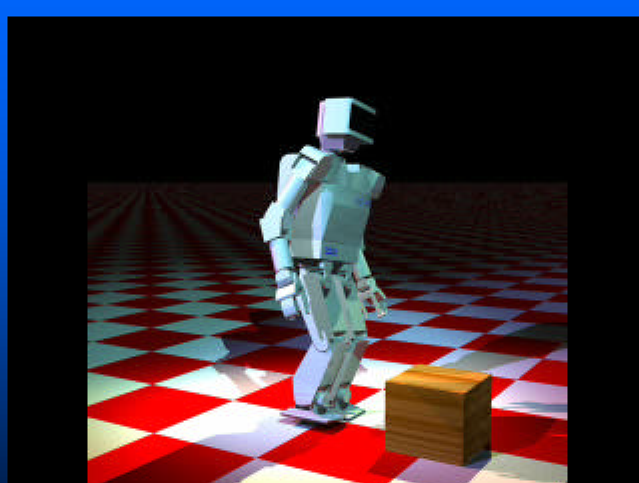


# Single-leg Example



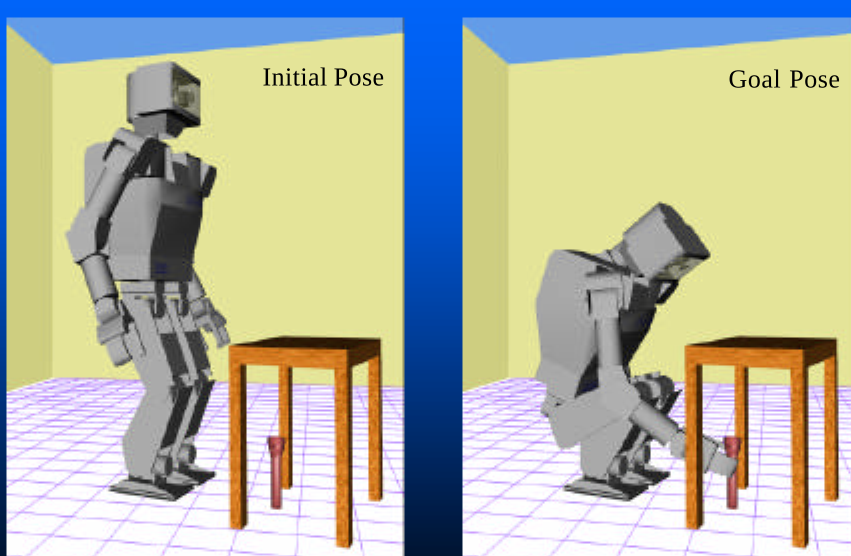




# Single-leg Example

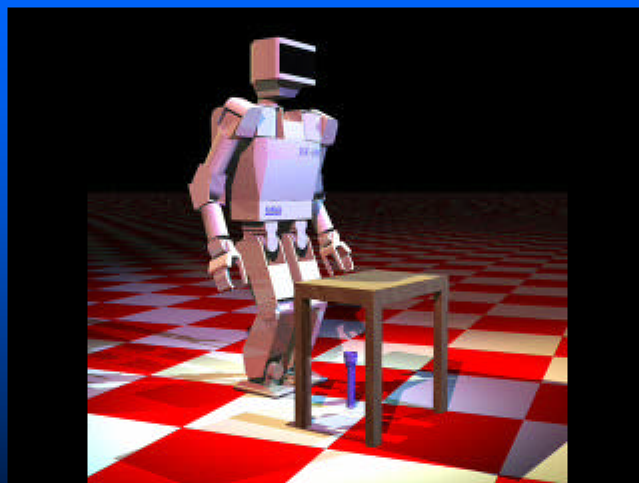


## Single-leg Example

## Dual-leg Example



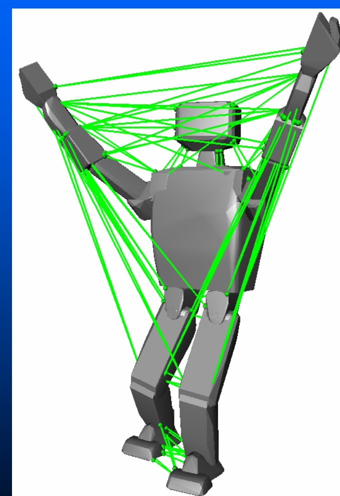
## Dual-leg Example



## Dual-leg Example

# Summary

- Efficient approach to self-collision detection brings self-collision testing closer to the servo control layer
- Extend method to handle non-convex polyhedral models
- Optimize the number of distance calculations using maximum joint velocity bounds
- Applications in multi-robot coordination and computer animation of articulated figures



# Future Work

- Bring Self-collision testing closer to the servo control layer
- Extend method to handle non-convex polyhedral models
- Optimize the number of distance calculations using maximum joint velocity bounds
- Can be used to prevent collisions between environment obstacles of known geometry
- Applications in multi-robot coordination and computer animation of articulated figures

## Collaborators

- Satoshi Kagami (AIST)
- Koichi Nishiwaki (U. Tokyo)
- Steve LaValle (Univ. of Illinois)
- Joel Chestnutt (CMU)
- Katsu Yamane (CMU & U. Tokyo)
- Hirochika Inoue (U. Tokyo)
- Masayuki Inaba (U. Tokyo)
- Takeo Kanade (AIST & CMU)
- Jean-Claude Latombe (Stanford)