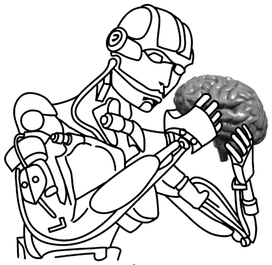
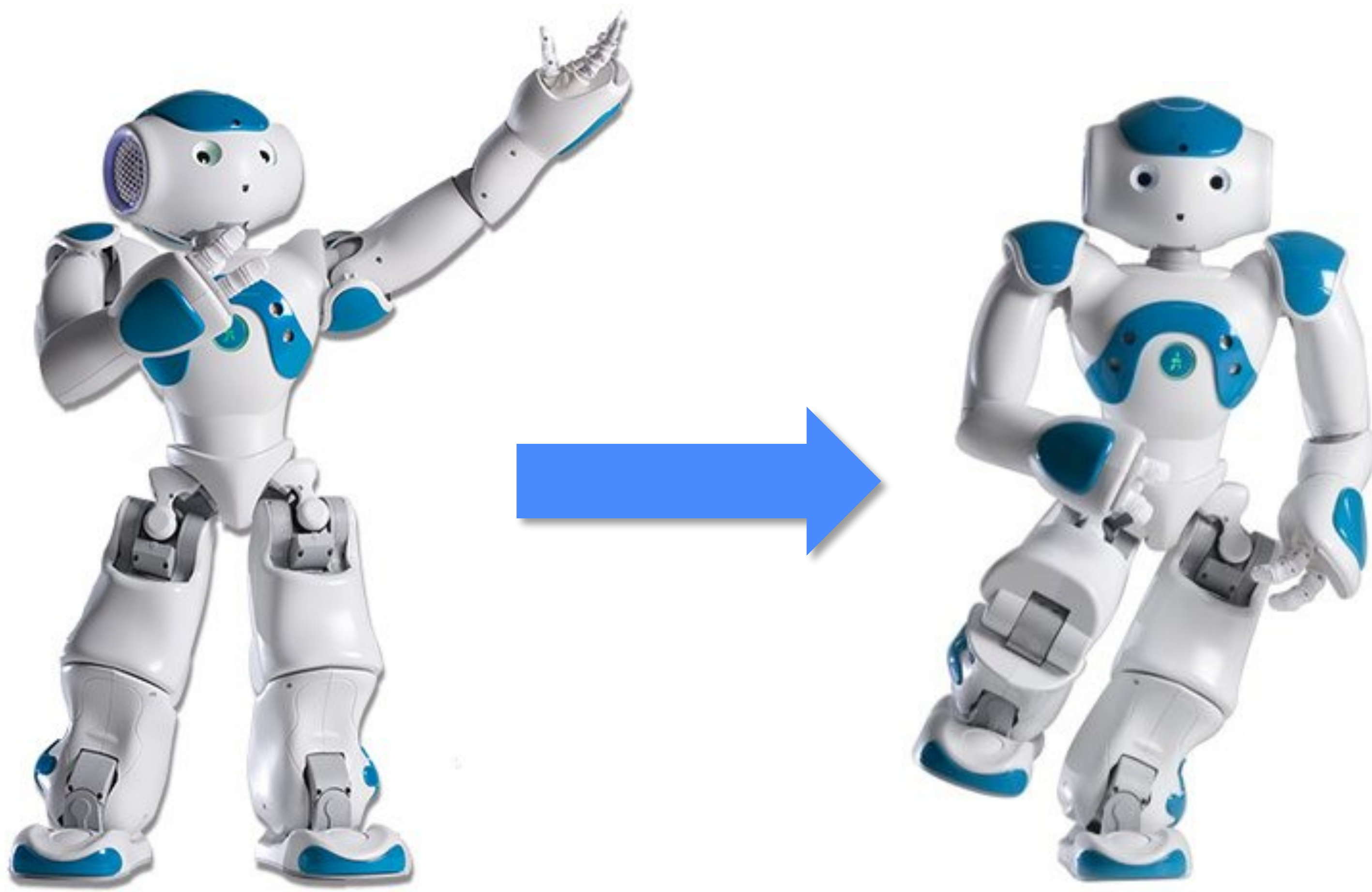
CS545—Project NAO

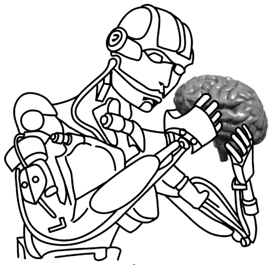


- Project Description
  - NAO standing on one leg
  - Move back-and-forth between both legs
  - Step in place
  - Optional: step forward
  - Optional: just do something new
- Basic Math of Approach
- Programming in the SL Simulator

Balancing on One Leg



Learn About NAO DOFs

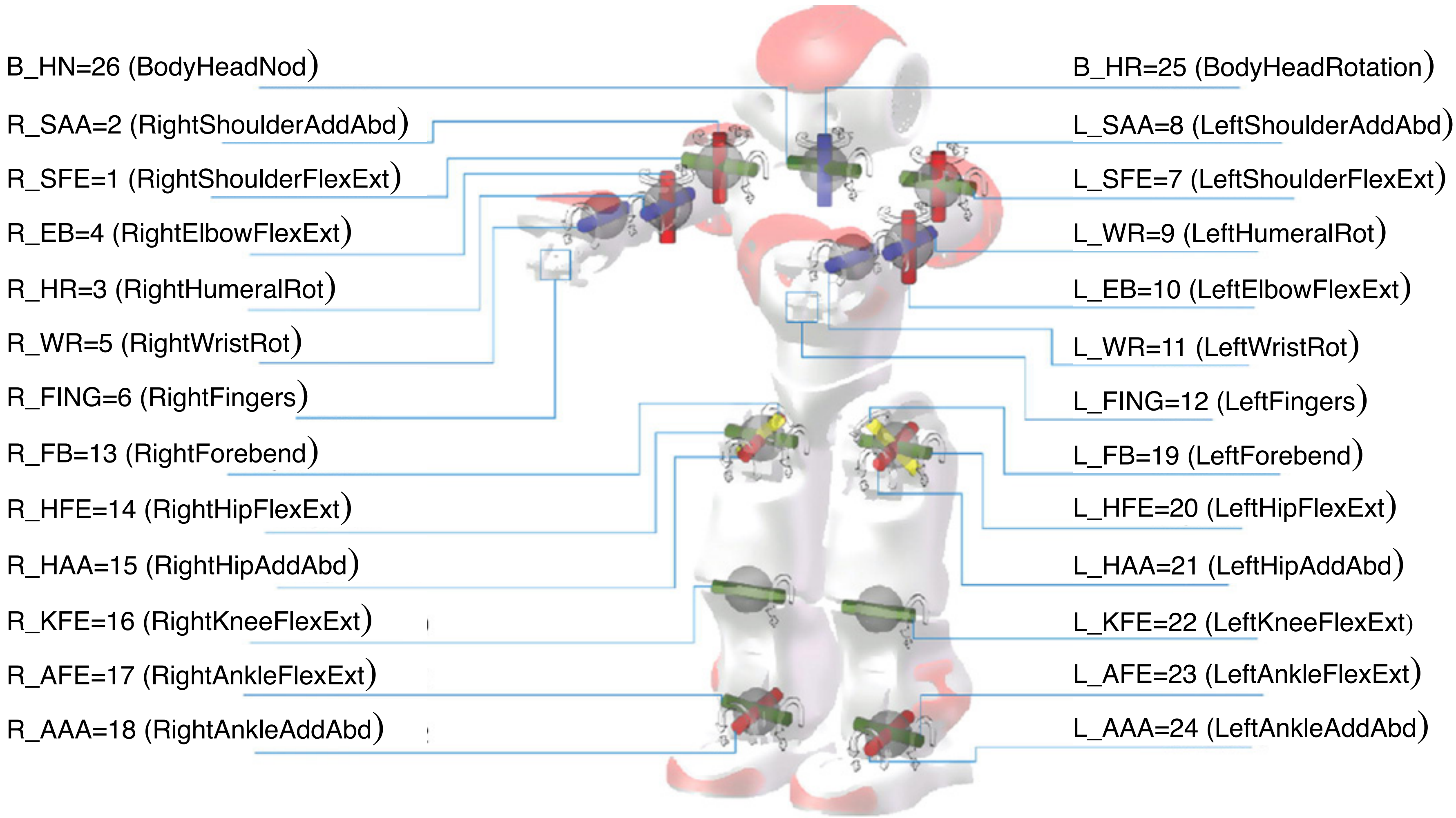


- Start NAO simulator (the robot hangs in the air and all DOFs can move freely)
- Use nao.task>where to see the current name, number, and value of all DOFs. The DOFs on the right is what you are going to need most
- Use nao.task>go and move these DOFs to new desired positions. Observe where the simulator moves to understand the DOFS
- Click on the Graphics Window to see a pop-up window how to change the view of the graphics

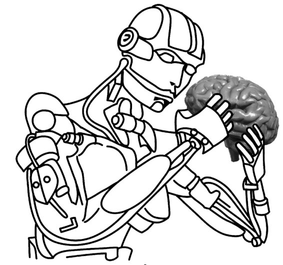
14: R_HFE
15: R_HAA
16: R_KFE
17: R_AFE
18: R_AAA

20: L_HFE
21: L_HAA
22: L_KFE
23: L_AFE
24: L_AAA

NAO DOF Definition in SL

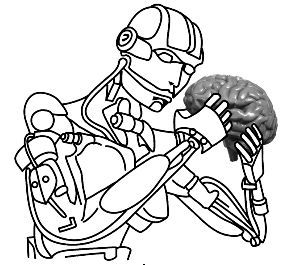


Basic Approach



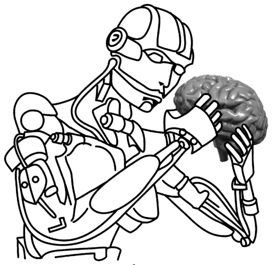
- Initial: Stand on both feet, maybe squat a bit
- Move Center of Gravity (COG) projection in the x-y plane to be in the middle of right foot
- Lift left foot up
- Put left foot down again, move COG projection to left foot, move right foot up
- Continuously stepping in place
- OPTIONAL: make small forward progress while stepping in place

Basis Functions You Need



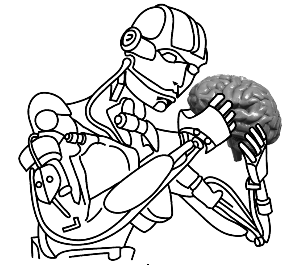
- Min jerk movements (or cubic spline), in either joint space or COG space (Homework 1 and 2)
- Inverse kinematics controller for COG (Homework 2) (COG position/velocity, COG Jacobian, and pseudoinverse will be provided)

Approach ONE (Simple, but very manual and hacky)



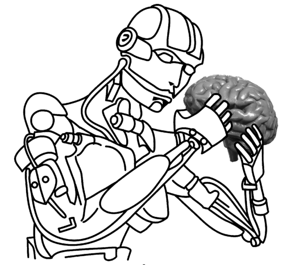
- Somehow find a joint space target for the robot to stand on one foot
  - Use nao.task>freezeBase to put the robot on the floor
  - Use nao.task>go to give individual joints desired targets
  - Observe the “red ball” on the floor moving to the center of the right foot
  - Do very small changes in joint angles, otherwise the robot falls over. Use nao.task>reset to put the robot back on the floor
  - Note that moving one leg alone creates a conflict between both legs, as they are coupled through a looped dynamics
- After you have an appropriate joint-space target, use a min-jerk movement (or cubic spline) to go there (like HW3), then you should be able to lift the left leg with a simple joint space movement
- Use nao.openGL>coordDisplay to visualize joint names (accept all defaults)

Approach TWO (Clean but more technical)



- Move COG to center of right foot by inverse kinematics
  - Use nao.task>where_cog to see a print-out of COG position
  - Use nao.task>cwhere to see a print-out of foot positions. This print-out will give you the target position of for the COG for moving over a foot
  - Plan min jerk (cubic spline) trajectory of COG position to move from current position to desired position
  - Execute with inverse kinematics
- Lift left foot up with simple joint space movement

Theory of COG Inverse Kinematics



$$\text{COG: } \mathbf{x}_{cog} = \frac{1}{\sum_{i=1}^{n} m_i} \sum_{i=1}^{n} m_i \mathbf{x}_{i,cog}$$

$$\text{COG Jacobian: } \mathbf{J}_{cog} = \frac{\partial \mathbf{x}_{cog}}{\partial \boldsymbol{\theta}} = \frac{1}{\sum_{i=1}^{n} m_i} \sum_{i=1}^{n} m_i \frac{\partial \mathbf{x}_{i,cog}}{\partial \boldsymbol{\theta}}$$

$$\text{Floating Base COG Jacobian: } \mathbf{J}_{cog,float} = \begin{bmatrix} \mathbf{J}_{cog} & \mathbf{J}_{base} \end{bmatrix}$$

$$\text{Constraints from standing on 2 feet: } \mathbf{J}_{feet,float} \begin{bmatrix} \dot{\boldsymbol{\theta}} \\ \dot{\mathbf{x}}_{base} \\ \boldsymbol{\omega}_{base} \end{bmatrix} = 0 \qquad \text{(no slipping)}$$

$$\text{Null Space Projection for Constraints: } \mathbf{N}_c = \left(\mathbf{I} - \mathbf{J}^{\#}_{feet,float} \mathbf{J}_{feet,float} \right)$$

$$\text{Constraint COG Jacobian: } \mathbf{J}_{cog,const} = \mathbf{J}_{cog} \mathbf{N}_c$$

Inverse Kinematics with Constraint COG Jacobian

- Given: Desired trajectory of COG

$$\mathbf{x}_{cog,des}, \dot{\mathbf{x}}_{cog,des}$$

- Reference COG velocity

$$\dot{\mathbf{x}}_{cog,ref} = k_p \left(\mathbf{x}_{cog,des} - \mathbf{x}_{cog} \right) + \dot{\mathbf{x}}_{cog,des}$$

- IK Solution

$$\begin{bmatrix} \dot{\boldsymbol{\theta}}_{des} \\ \dot{\mathbf{x}}_{base,des} \\ \boldsymbol{\omega}_{base,des} \end{bmatrix} = \mathbf{J}^{\#}{}_{cog,const} \dot{\mathbf{x}}_{cog,ref} \qquad \boldsymbol{\theta}_{des}(t+1) = \dot{\boldsymbol{\theta}}_{des}(t)\Delta t + \boldsymbol{\theta}_{des}(t)$$

Implentation In SL

- balance_task.cpp is the skeleton to use
- All important variables are pre-computed and commented