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

B_HN=26 (BodyHeadNod)

R_SAA=2 (RightShoulderAddAbd)
R_SFE=1 (RightShoulderFlexExt)
R_EB=4 (RightElbowFlexExt)
R_HR=3 (RightHumeralRot)
R_WR=5 (RightWristRot)
R_FING=6 (RightFingers)
R_FB=13 (RightForebend)
R_HFE=14 (RightHipFlexExt)
R_HAA=15 (RightHipAddAbd)
R_KFE=16 (RightKneeFlexExt)
R_AFE=17 (RightAnkleFlexExt)
R_AAA=18 (RightAnkleAddAbd)

B_HR=25 (BodyHeadRotation)

L_SAA=8 (LeftShoulderAddAbd)
L_SFE=7 (LeftShoulderFlexExt)
L_WR=9 (LeftHumeralRot)
L_EB=10 (LeftElbowFlexExt)
L_WR=11 (LeftWristRot)
L_FING=12 (LeftFingers)
L_FB=19 (LeftForebend)
L_HFE=20 (LeftHipFlexExt)
L_HAA=21 (LeftHipAddAbd)
L_KFE=22 (LeftKneeFlexExt)
L_AFE=23 (LeftAnkleFlexExt)
L_AAA=24 (LeftAnkleAddAbd)
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 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

\[ \text{COG: } x_{\text{cog}} = \frac{1}{\sum_{i=1}^{n} m_i} \sum_{i=1}^{n} m_i x_{i,\text{cog}} \]

\[ \text{COG Jacobian: } J_{\text{cog}} = \frac{\partial x_{\text{cog}}}{\partial \theta} = \frac{1}{\sum_{i=1}^{n} m_i} \sum_{i=1}^{n} m_i \frac{\partial x_{i,\text{cog}}}{\partial \theta} \]

\[ \text{Floating Base COG Jacobian: } J_{\text{cog, float}} = \begin{bmatrix} J_{\text{cog}} & J_{\text{base}} \end{bmatrix} \]

Constraints from standing on 2 feet: \[ J_{\text{feet, float}} \begin{bmatrix} \dot{\theta} \\ \dot{x}_{\text{base}} \\ \omega_{\text{base}} \end{bmatrix} = 0 \quad \text{(no slipping)} \]

Null Space Projection for Constraints: \[ N_c = \left( I - J_{\text{feet, float}}^\# J_{\text{feet, float}} \right) \]

Constraint COG Jacobian: \[ J_{\text{cog, const}} = J_{\text{cog}} N_c \]
Inverse Kinematics with Constraint COG Jacobian

• Given: Desired trajectory of COG
  \[ \mathbf{x}_{\text{cog}, \text{des}}, \dot{\mathbf{x}}_{\text{cog}, \text{des}} \]

• Reference COG velocity
  \[ \dot{\mathbf{x}}_{\text{cog}, \text{ref}} = k_p \left( \mathbf{x}_{\text{cog}, \text{des}} - \mathbf{x}_{\text{cog}} \right) + \dot{\mathbf{x}}_{\text{cog}, \text{des}} \]

• IK Solution

\[
\begin{bmatrix}
\dot{\theta}_{\text{des}} \\
\dot{\mathbf{x}}_{\text{base}, \text{des}} \\
\omega_{\text{base}, \text{des}}
\end{bmatrix} = J^\#_{\text{cog, const}} \dot{\mathbf{x}}_{\text{cog}, \text{ref}} \quad \theta_{\text{des}}(t + 1) = \dot{\theta}_{\text{des}}(t) \Delta t + \theta_{\text{des}}(t)
\]
Implementation In SL

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