Projects From Last Quarter
Robot models

Arms

- KUKA IIWA
- Franka Emika Panda
- Panda + Gripper
Robot models

Mobile bases

- Ocean One
- Mobile base + 1 Panda
- Mobile base + 2 Panda
- Flying base + 1 arm
- Flying base + 2 arm
Sensors

- Sensors: force sensor, pseudo-camera, joint encoders
- Add noise, read/write through redis-server (batch command)
Simulation Worlds
General Workflow

1. Create .obj + .mtl files for object meshes (CAD, Blender)
2. Select robot urdf and add link/joint modifications (attaching tool)
3. Create world urdf files for simulation description
4. Setup simulation.cpp file for world specification and sensor information
5. Setup controller.cpp file for state machine and control algorithms
6. Debugging with redis

Most information can be obtained from “SAI.URDF.Instructions” in Canvas
.obj and .mtl Workflow

- Export CAD to .obj or obtain mesh file (.ply, .obj)
- Use Blender to import mesh file, re-center object, modify object, add textures
- Export modified object as *.obj and *.mtl objects
- Option to use V-HACD plug-in to create convex decomposition
  - You should only use this if a primitive collision mesh does not work for your application
- Stanford Bunny example
Collision Demo
Robot URDF

- Robot urdf and mesh files are given (panda, mobile base, toro, quadcopter)
- Possible modifications include attaching tool to tip of the arm
  - Adjusting torque/velocity/motion limits
  - Add fixed joint connecting last link to tool link, with custom mesh for tool link
  - Add grippers with additional prismatic joint and gripper link
- Reference urdf (Panda with attached plate)
World URDF File

- World urdf file specifies robots and objects placement
- Specify robots, static objects, gravity, object centering in the world
  - Static objects: objects that are fixed to the ground
  - Robots: robotic platform, any manipulated object
- Collision demo example
Simulation File

- **Highlight key points:**
  - Static and kinetic friction
  - Coefficient of restitution (typically 0)
  - Redis keys setup
  - Graphics update
  - Function declaration

- **Main loop should only be running simulation integration (> 1 kHz)**
  - If you have more complicated simulation worlds, you will need to slow down the simulation
  - Batch write/read to the redis-server (~1 ms)
Controller File

• Highlight key points:
  ○ Read keys from redis from simulation
  ○ Run through state machine logic
  ○ Use SAI2-Primitives library for controllers (Pos-Ori, etc.): examples from SAI2-Examples
  ○ Primitives use OTG library (Online Trajectory Generation)

• Main loop should be running > 1 kHz
  ○ Avoid long blocking code segments
  ○ Avoid dynamic memory allocation
  ○ Run estimators at a slower frequency
Debug Tips

● Don’t make the environment complicated for robust simulation
  ○ Keep the collision mesh simple - don’t want to slow down simulation
● Use the collision mesh as the visual mesh first, then replace with visual mesh
● Don’t start objects initially touching - collision will be disabled
● Use redis-server in separate terminal to read published data
  ○ Use the batch redis-server publish/read
  ○ State machine state, control torques, force sensors, etc