Algorithmic Human-Robot Interaction

Language in A-HRI (Papers)
System Design Workshop

CSCI 7000

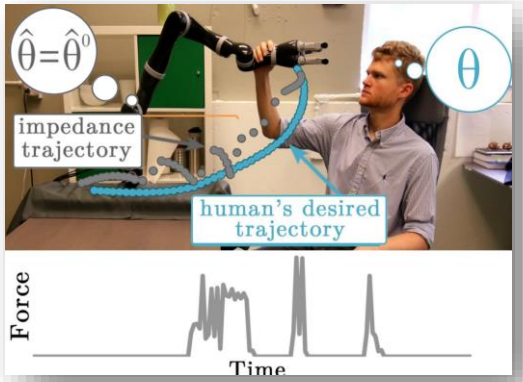
Prof. Brad Hayes

University of Colorado Boulder

Papers for Thursday 3/7:
Interpreting and Expressing Goals
(E-mail Bradley.Hayes@Colorado.edu to sign up)

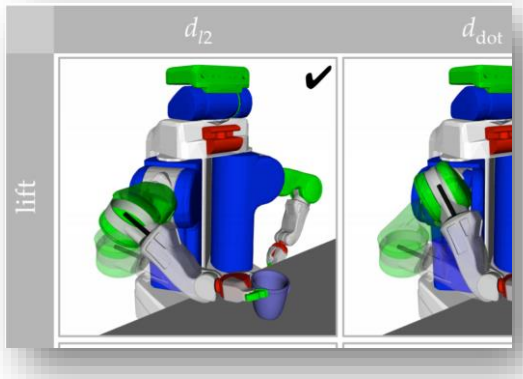
Learning Robot Objectives from Physical Human Interaction – Bajcsy et al.

Pro: Con:



Expressing Robot Incapability – Kwon et al.

Pro: Con:



Quiz 5

(5 Minutes)

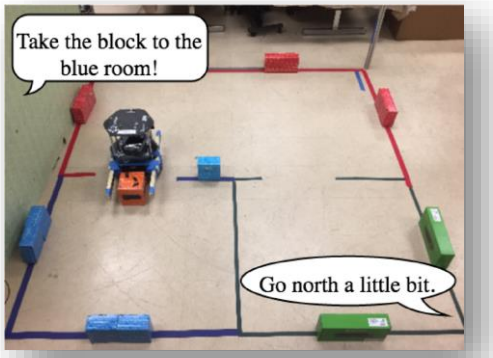
Papers for Thursday 2/28:
Natural Language Understanding

Robust Robot Learning from Demonstration and Skill Repair Using Conceptual Constraints – Mueller et al.

Pro: Jack Kawell Con: Matthew Luebbers

Accurately and Efficiently Interpreting Human-Robot Instructions of Varying Granularities – Arumugam et al.

Pro: Karthik Palavalli Con: Ian Loefgren



System Design Workshop (Part I)

Spend the remainder of class defining the major components of your final project

(To be presented and discussed Tuesday)

Deliverable: Block diagram with details for each component

(Powerpoint slide is generally easiest mode of presentation)

0: Preliminaries

Set up your source control repo
(add me as a collaborator: hayesbh)

Document all the libraries you pull in for your project (and versions)

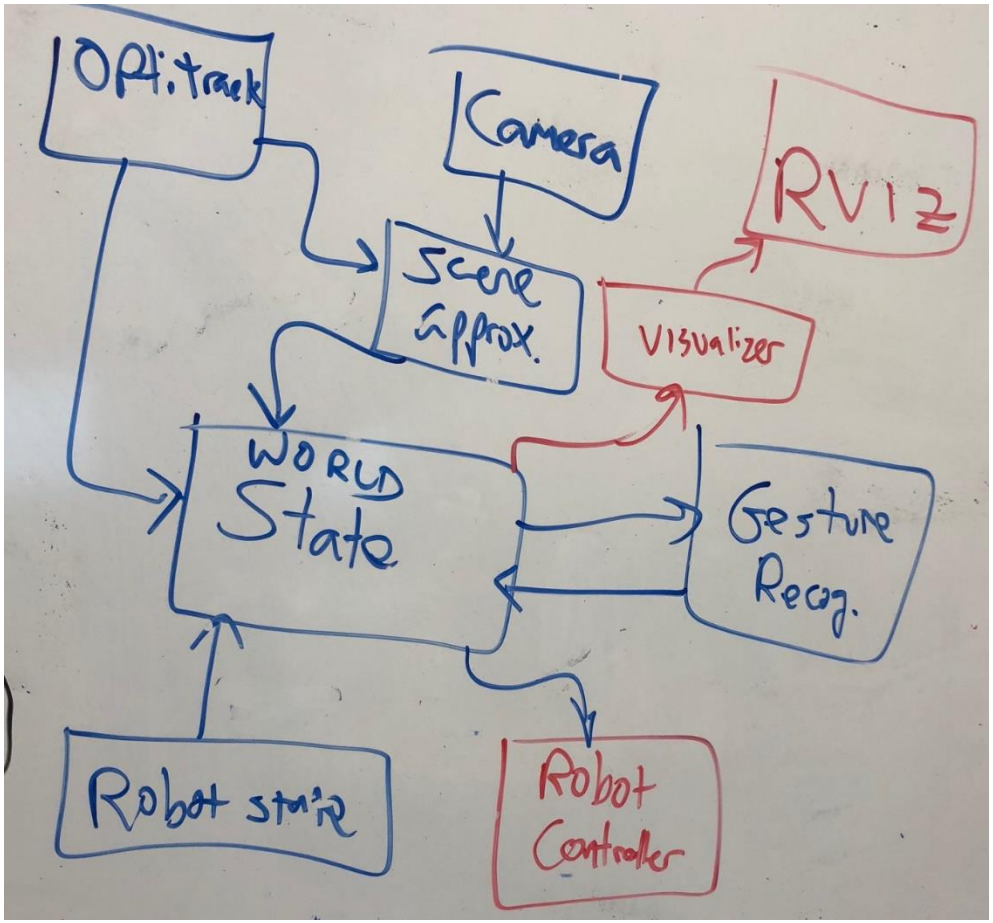
Don't commit these,
just list them in your README's "Setup Instructions"

Read the snarky PDB tutorial:
https://github.com/spiside/pdb-tutorial

1: Designing Your Modular System

High level design chart

- Each box is an executable or sensor
- Show interactions between components (with direction)
- Be able to justify each arrow
- Don’t worry about the **how** or any **existing work**



2: Specifying Node Functions

For each box in your chart:

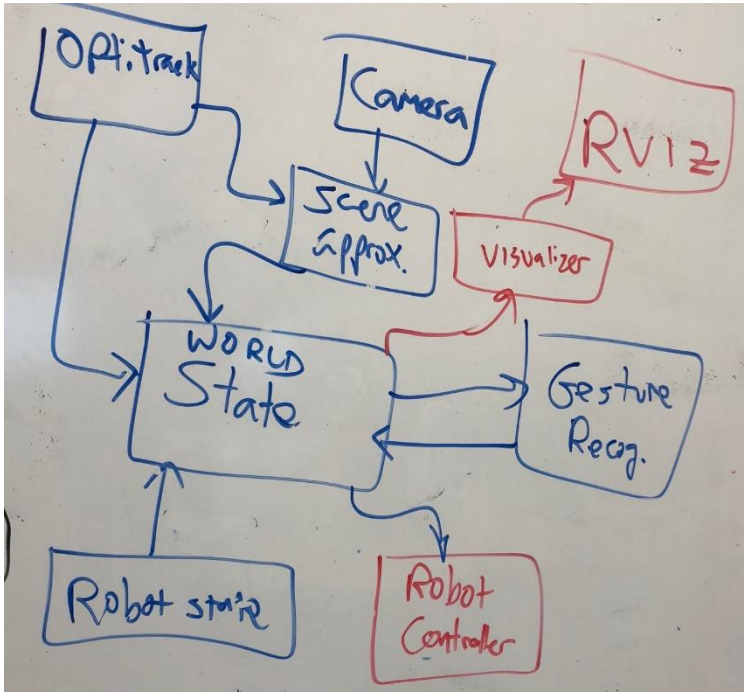
- What does it do? (1-2 lines)
- What information would it need to perform its function? (High level description)

World State:
Aggregates and synchronizes all world state information

Needs postprocessed inputs from all sensors

Scene Approximator:
Simplifies world's objects into a list of primitive objects (boxes, cylinders, etc.) and their poses.

Needs all visual/tracking inputs from the world.



3: Making Inputs/Outputs Explicit

For each outbound edge:

- What data is it outputting?
- Is this the right component to provide this information? (How close to 'source' is it?)

For each inbound edge:

- Where's the data from?
- What data are you expecting?

Scene Approximator:

Outputs on /simplified_objects:

Array of simple_objects (new msg):

int (Object ID)

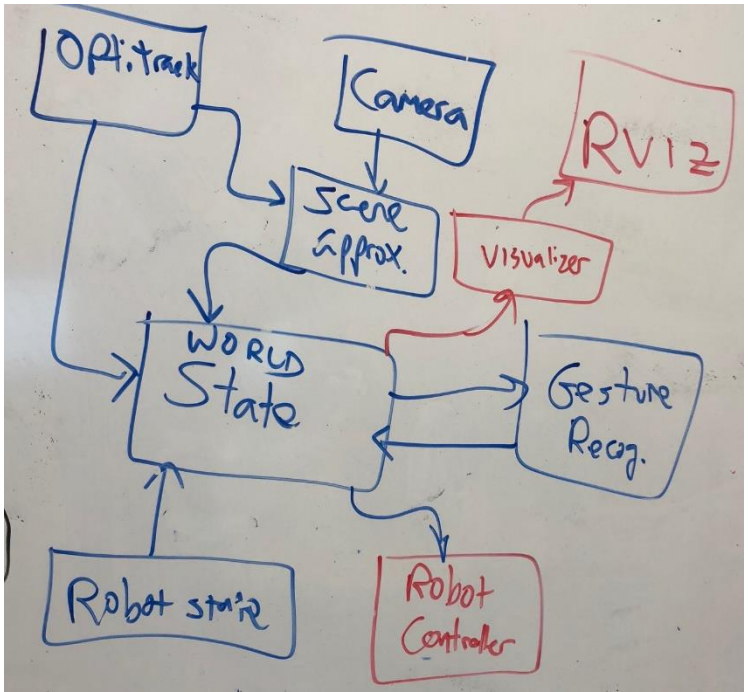
int (Object Type)

geometry_msgs/Pose (Object Pose)

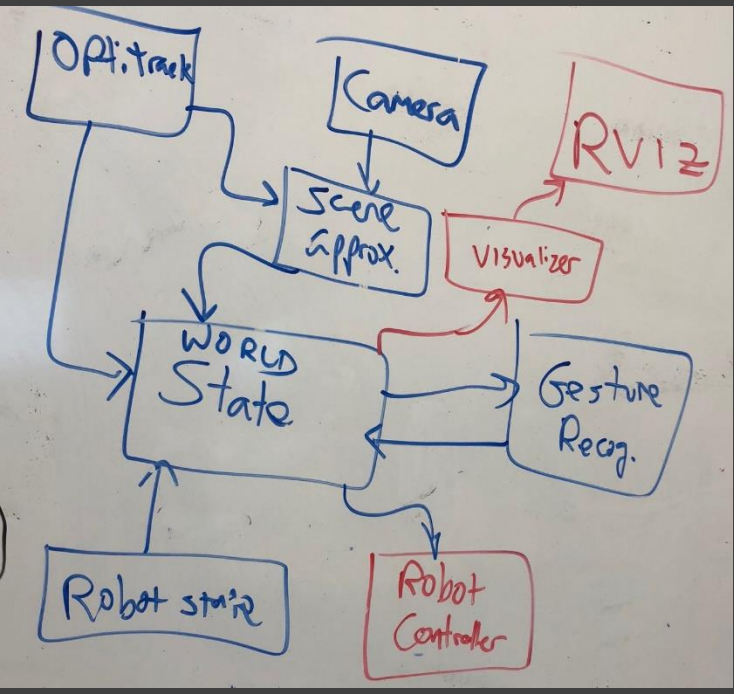
Inputs:

Camera: sensor_msgs/Image (Rectified camera image) on /camera/image_raw

Optitrack: geometry_msgs/PoseStamped (Object poses) on /optitrack/pose



4: Outlining Node Functionality

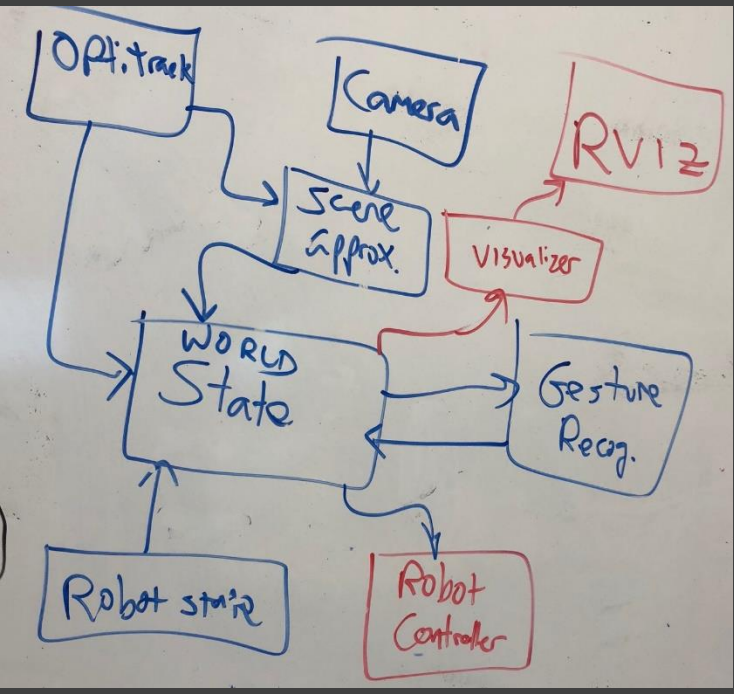


For each node that you're building:

- Define subscriber callbacks
  - Is processing happening now? Or elsewhere.
  - Where is the data being stored?
- Define major classes:
  - Does it need to be a class?
- Define major functions:
  - Write empty function definitions for major processing needs

    ```
    def my_function(args):
        pass
    ```
  - Write dummy functions to send empty/test data on all publishers.
- For each major function:
  - Write pseudocode comments for every step
  - Iterate to add more comments/specificity
  - When done, you should be able to turn this over to anyone in the class to implement!

5: Implementation and Debugging



Set up test data to run through the system!

- Figure out how you'll know your component works

Set up test output to come from your system!

- Tests need not be exhaustive, just enough to know that things are working.

Turn your comments into real code:

Commit often!

- If something breaks, document it in your commit
  - Don't hide from committing until its fixed!
- Small steps are valuable
- Use informative commit messages

Fail early and loudly

- Bradley.Hayes@Colorado.edu

1. Set up Git repo
2. Draw your system diagram
3. Describe your nodes' purposes
4. What are your inputs/outputs?
   - Group into ROS topics/message types
5. Outline your nodes' functionality
   - Increasingly descriptive pseudocode
6. Start turning pseudocode into real code

Summary