ROS-Industrial Basic Developer's Training Class

October 2024

Southwest Research Institute

Session 4:
Motion Planning

Moveit! Planning using C++

Intro to Planners

Intro to Perception

Southwest Research Institute

Motion Planning in C++

MoveIt! provides a high-level C++ API:

moveit_cpp

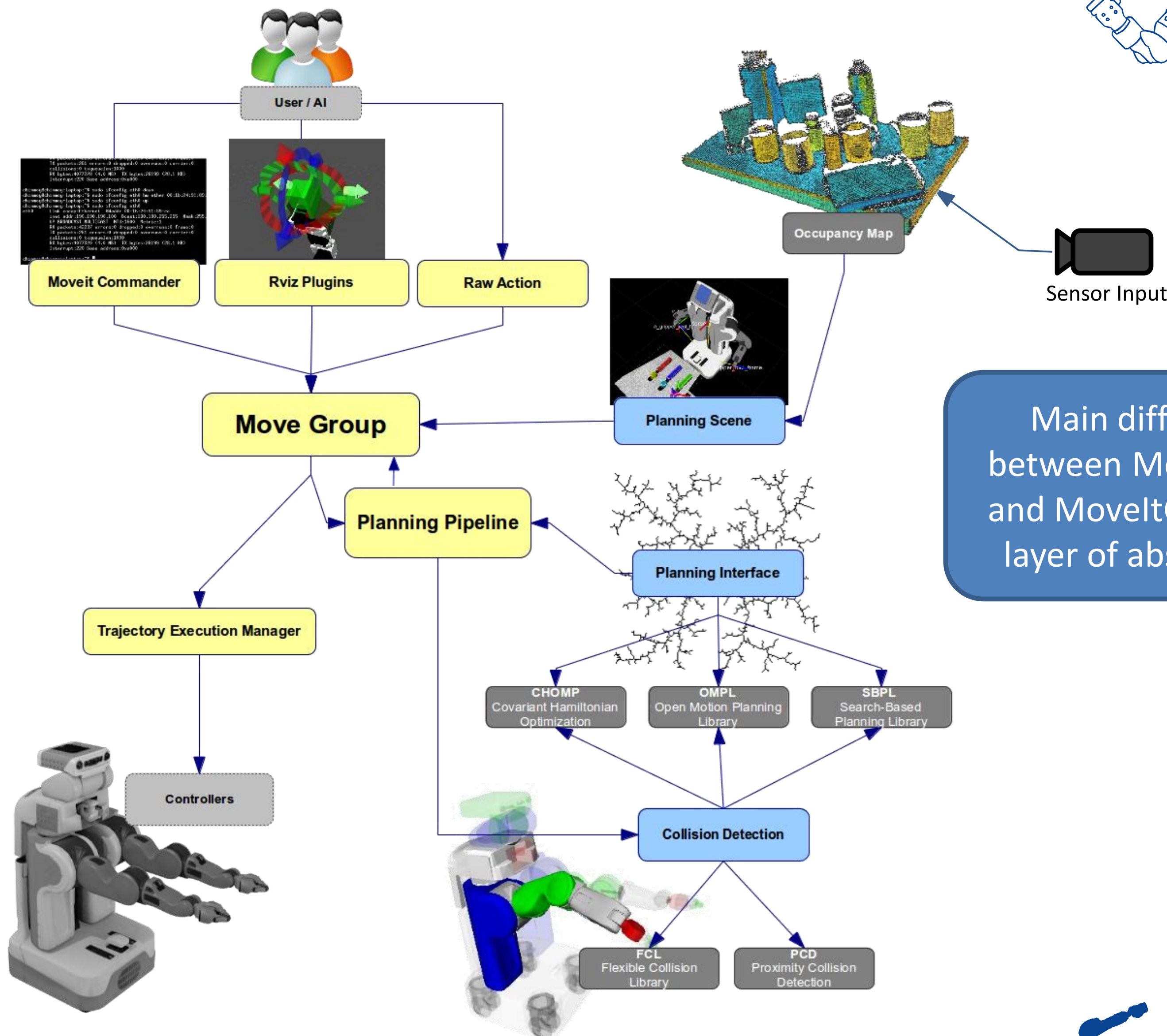
```
#include <moveit/moveit_cpp/moveit_cpp.h>
...
moveit_cpp::MoveItCpp::Ptr moveItCpp = make_shared(node);
moveit_cpp::PlanningComponent::Ptr planner = make_shared(“arm", moveItCpp);

planner->setGoal(“home”);
planner->plan();
planner->execute();
```

5 lines = collision-aware path planning & execution

Amazing!

Reminder: MoveIt! Complexity



Main difference between MoveGroup and MoveItCpp is the layer of abstraction

http://moveit.ros.org/wiki/High-level_Overview_Diagram
http://moveit.ros.org/wiki/Pipeline_Overview_Diagram

Motion Planning in C++

Pre-defined position:

```
planner.setGoal("home");
```

Joint position:

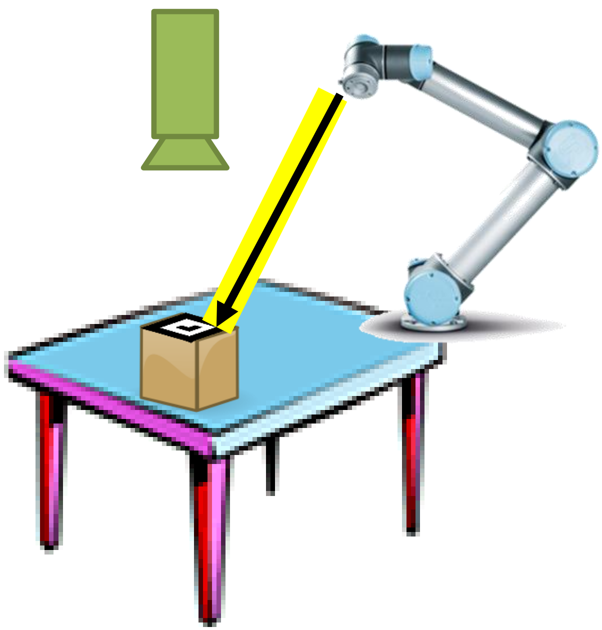
```
robot_state::RobotState joints.setStateValues(names, positions);
planner.setGoal(joints);
```

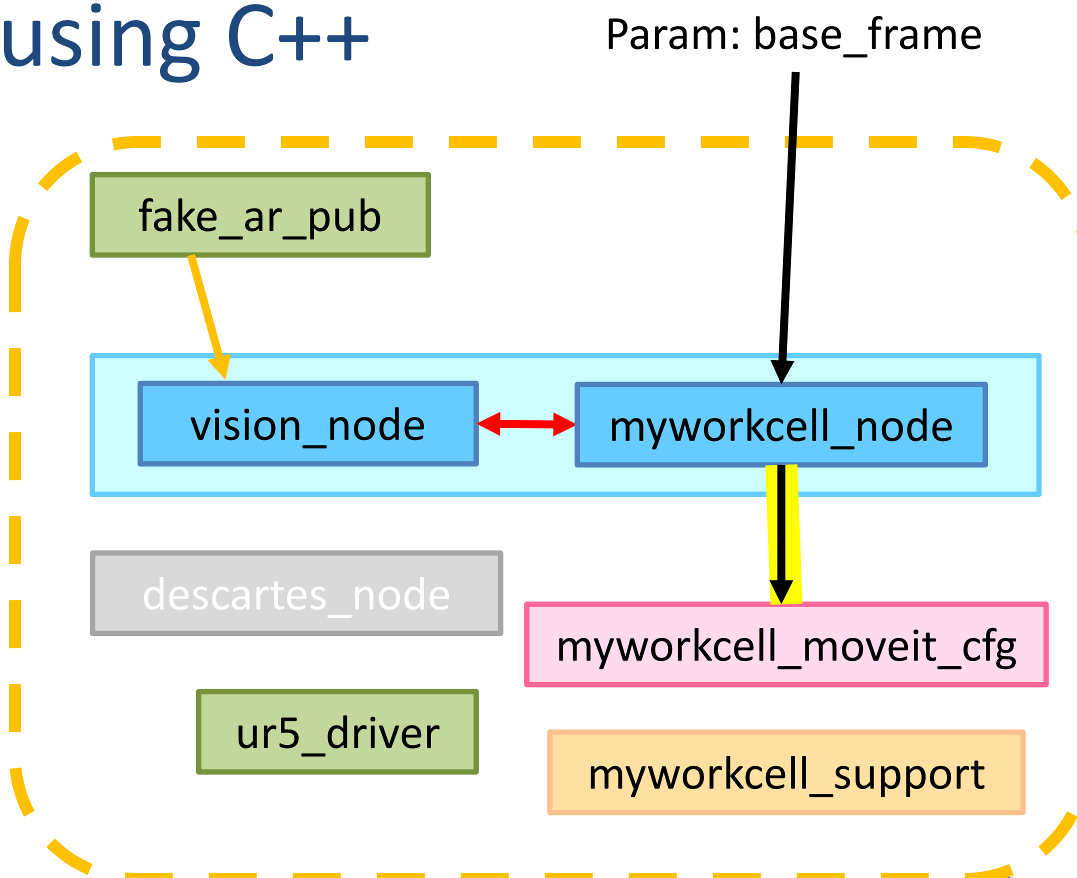
Cartesian position:

```
Affine3d pose = {x, y, z, r, p, y};
planner.setGoal(pose);
```

Exercise 4.0:

Motion Planning using C++





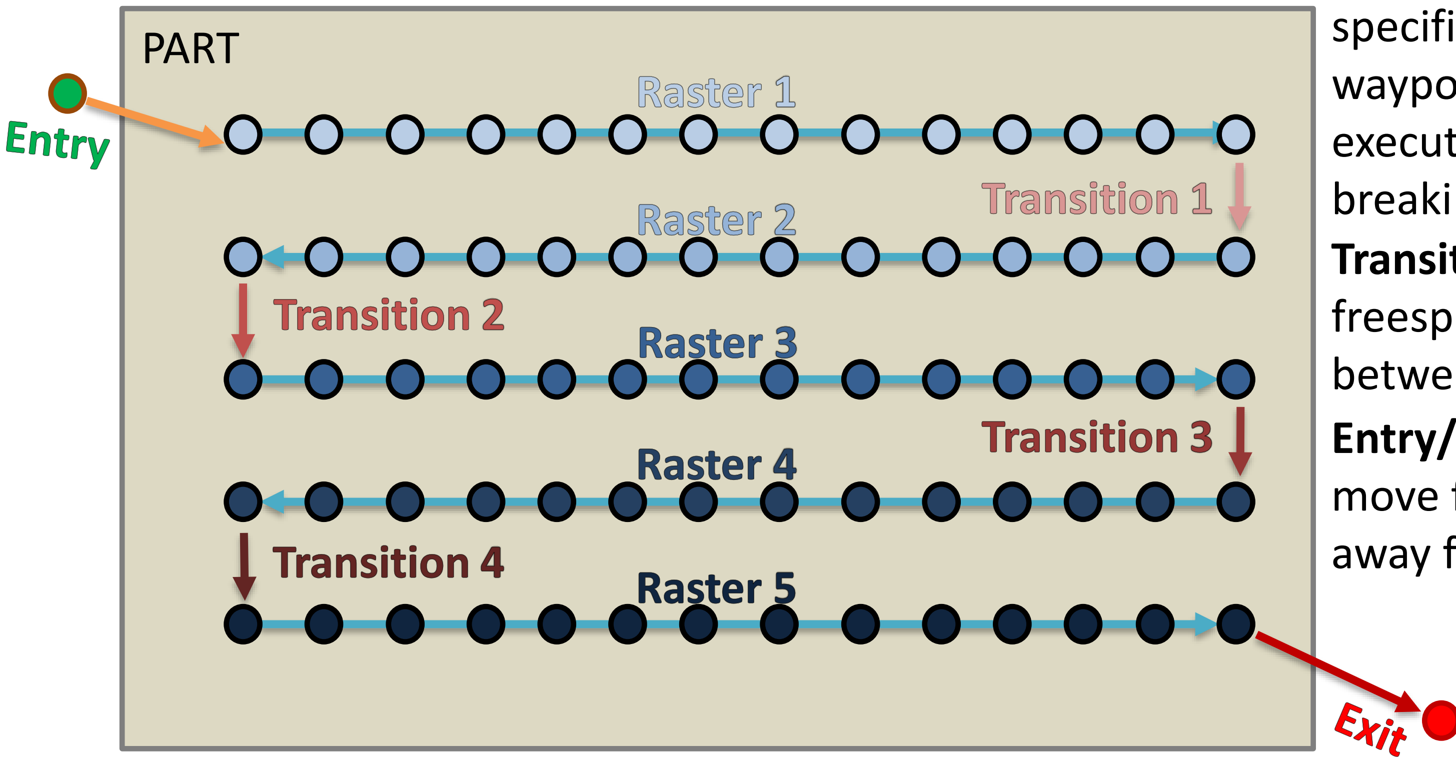
Intro to Planners

- Types of Motion Plans
- Basic Toolpath Plan
- Planning Workflows
- Common Motion Planners
  - OMPL
  - Descartes Light
  - TrajOpt
- Motion Planning Frameworks
- Simple Planning Pipelines
- Advanced Planning Pipelines

Types of Motion Plans

Freespace	Process	Combined
Motion plans between far-spaced start and end points	Motion plans optimize robot pose between under-constrained waypoints	Motion plans that can be segmented into portions that are freespace motions and others that are process motions
Example: Moving from a generic, off-the-surface "start pose" to the upper righthand corner of a surface for painting	Example: A continuous line mapped around the edge of a piece to be welded	Example: Moving from a generic, off-the-surface "start pose" to the edge of a jig-held part and then welding the edge at a known EE angle

Toolpath Plan Example



Definitions

Raster - A series of specified Cartesian waypoints to be executed without breaking*

Transition - A freespace move between rasters

Entry/Exit - A freespace move from/to a position away from the part

*depends on application

Common Motion Planners

Motion Planner	Application Space	Notes
OMPL	Free-space Planning	Stochastic sampling; Easy and convenient interface
TrajOpt	Trajectory Optimization	Optimize existing trajectory on constraints (distance from collision, joint limits, etc.)
Descartes Light	Cartesian path planning	Globally optimum; sampling-based search; Captures “tolerances”
Simple Planner	Free-space Planning	Naive simple linear interpolation between waypoints
STOMP	Free-space Planning	Optimization-based; Emphasizes smooth paths
CHOMP	Trajectory Optimization	Gradient-based trajectory optimization for collision avoidance and cost-reduction

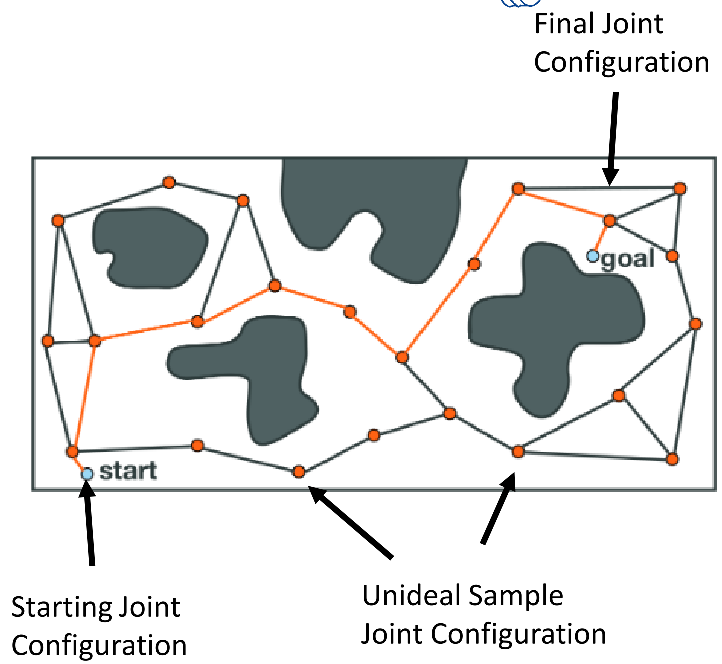
OMPL

Open Motion Planning Library:

Randomly Sample Valid Joint States then Solve for Sequence

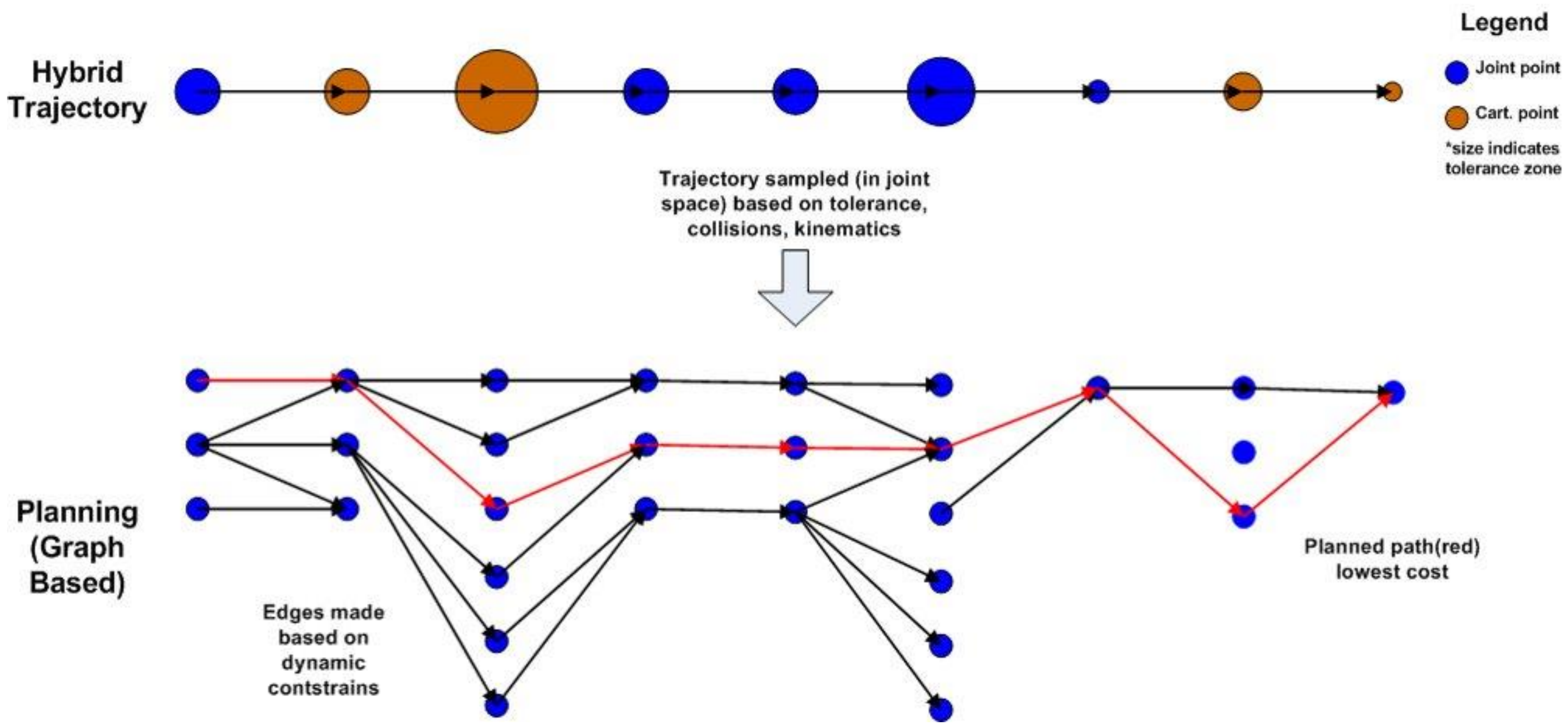
Planners we often use:

- RRT
  - Build a tree along different potential joint configurations to arrive at the final pose
- RRT-Connect
  - Build a tree from each side and try to *connect* them
  - Parameters
    - Range (same as above)
- See more at https://ompl.kavrakilab.org/planners.html



Descartes Light

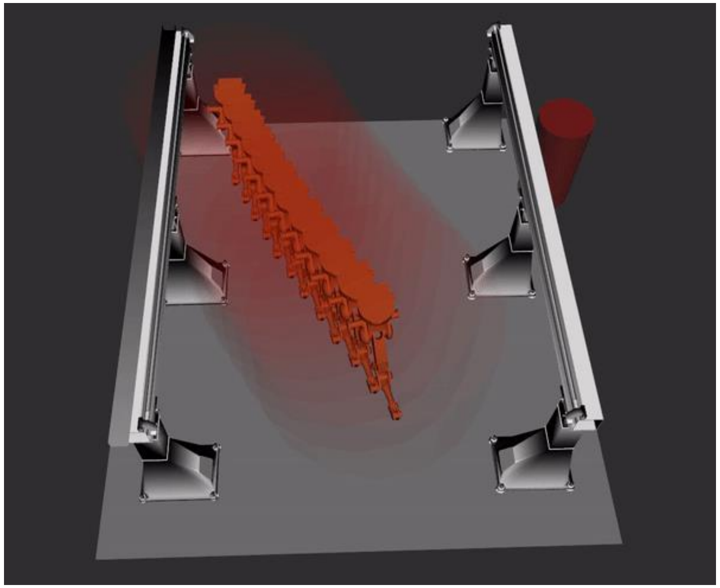
Sample 'all' Possible Solutions then Graph Search for Best Trajectory



TrajOpt

Optimize Seed Trajectory based on Weighted Cost Functions (distance from collision, joint limits, etc.)

- All parameters have a coefficient that can be increased/decreased to change its influence
- Example costs:
  - Proximity to a singularity
  - Velocity/Acceleration/Jerk smoothing
  - Avoid collisions
    - Weighed sums of all collision terms
    - Safety margin-based cost
  - Encourage/discourage DOF usage
    - Cartesian: rotation about z encouraged & unconstrained
    - Joint: usage of the wrist discouraged with a high cost
- Constraints are simply infinite costs
  - The absolute limit of the safety margin would be set and anything in collision with it would cause the planner to fail



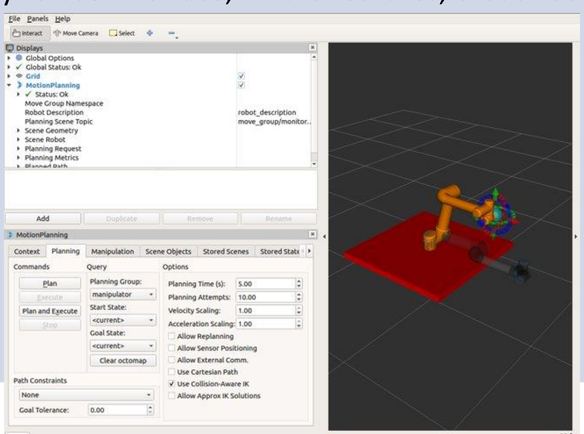
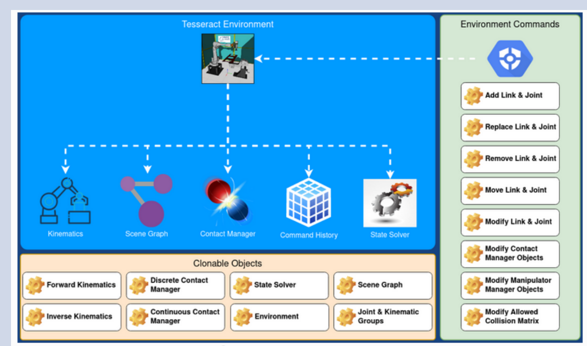
Motion Planning Environments

Interfaces used to generate motion plans can be:

- Open Source or License-based
- UI or script based
- Leverage a variety of planners
- Contain additional hooks to simulation packages

These differ from raw planners with:

- ROS API
- Collision environment management
- Visualization packages
- Planning pipeline/Task Constructor capabilities

MoveIt!/MoveIt!2	Tesseract
Easy to use interface, wizard features, broad toolset 	Enables very complex planning, different toolset 

Freespace Planners

Freespace

- Find Valid Path
- Smooth
- Generate Timestamps

Input

- Start & End States
- Seed Trajectory
- Seed Trajectory

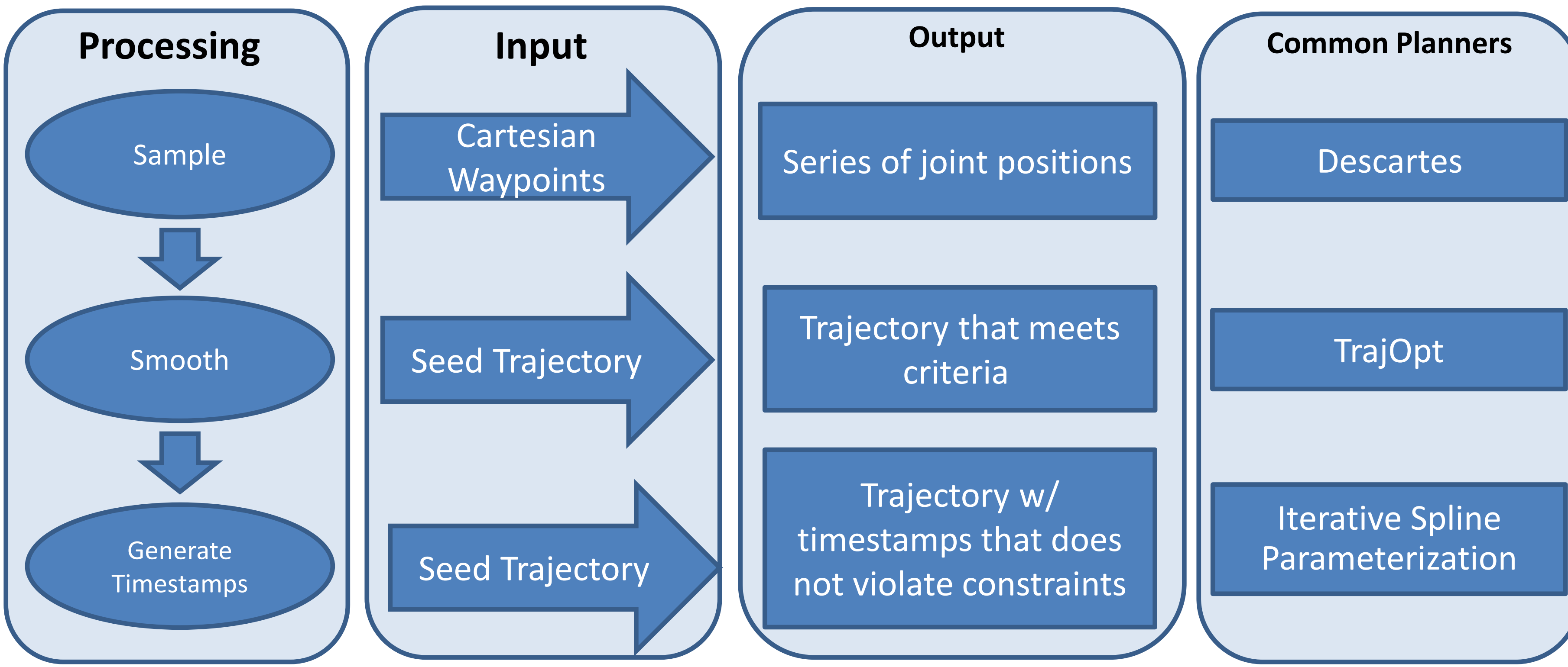
Output

- Valid trajectory
- Trajectory that meets criteria
- Trajectory w/ timestamps that does not violate constraints

Common Planners

- TrajOpt/OMPL
- TrajOpt
- Iterative Spline Parameterization

Processing Planners



INTRODUCTION TO PERCEPTION

Outline

- Camera Calibration
- 3D Data Introduction
- Explanation of the Perception Tools Available in ROS
- Intro to PCL tools
  - Exercise 4.1

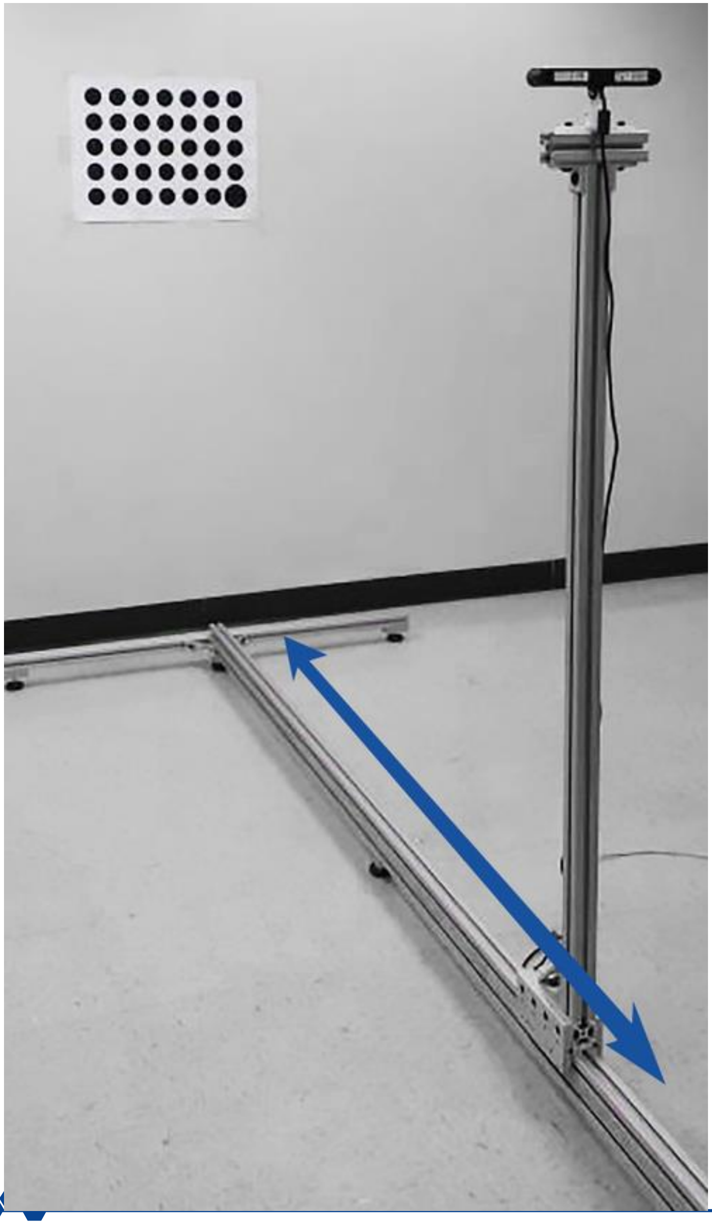
Objectives

- Understanding of the calibration capabilities
- Experience with 3D data and RVIZ
- Experience with Point Cloud Library tools*

Industrial Calibration

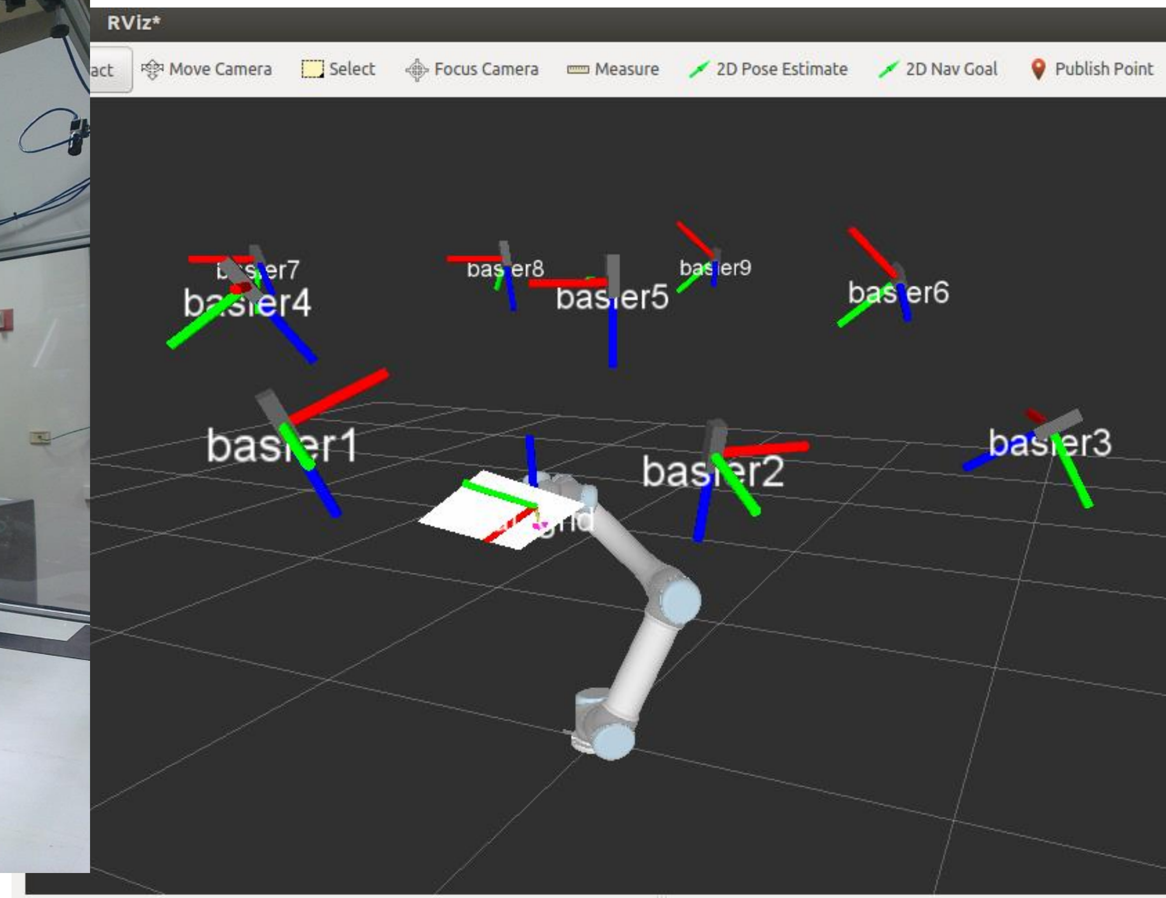
- Perform intrinsic and extrinsic calibration
- Continuously improving library
- Resources, library
  - Github link
  - Wiki link
- Resources, tutorials
  - Github industrial calibration tutorials link

Industrial (Intrinsic) Calibration



- The INTRINSIC Calibration procedure requires movement of the camera to known positions along an axis that is approximately normal to the calibration target.
- Using the resulting intrinsic calibration parameters for a given camera yields significantly better extrinsic calibration or pose estimation accuracy.

Industrial (Extrinsic) Calibration



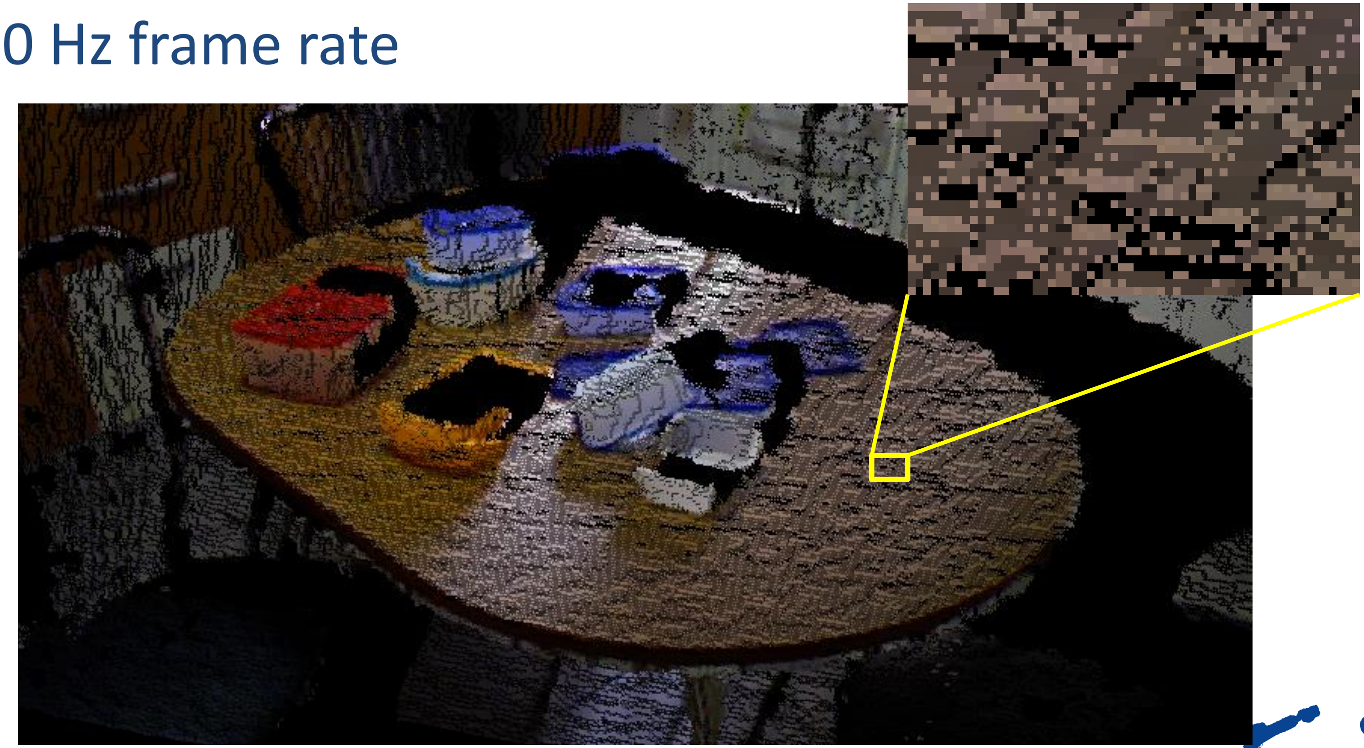
https://www.youtube.com/watch?v=MJFtEr_Y4ak

3D Cameras

- RGBD cameras, TOF cameras, stereo vision, 3D laser scanner
- Driver for Intel Realsense is available on the debian build farm
- ROS 2 driver for photoneo is community developed on github
- https://rosindustrial.org/3d-camera-survey

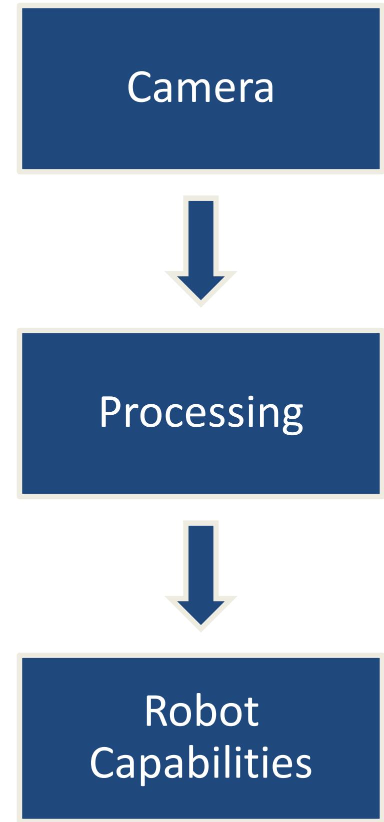
3D Cameras

- Produce (colored) point cloud data
- Huge data volume
  - Over 300,000 points per cloud
  - 30 Hz frame rate



Perception Processing Pipeline

- Goal: Gain knowledge from sensor data
- Process data in order to
  - Improve data quality ➡ filter noise
  - Enhance succeeding processing steps ➡ reduce amount of data
  - Create a consistent environment model ➡ Combine data from different view points
  - Simplify detection problem ➡ segment interesting regions
  - Gain knowledge about environment ➡ classify surfaces



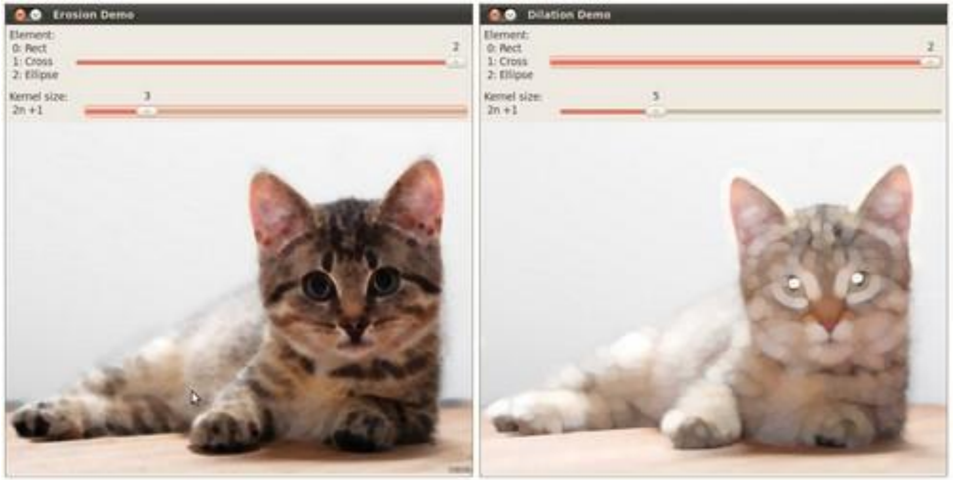
Perception Tools

- Overview of OpenCV
- Overview of PCL
- PCL and OpenCV in ROS
- Other libraries

- Focus on PCL tools for exercise

Perception Libraries (OpenCV)

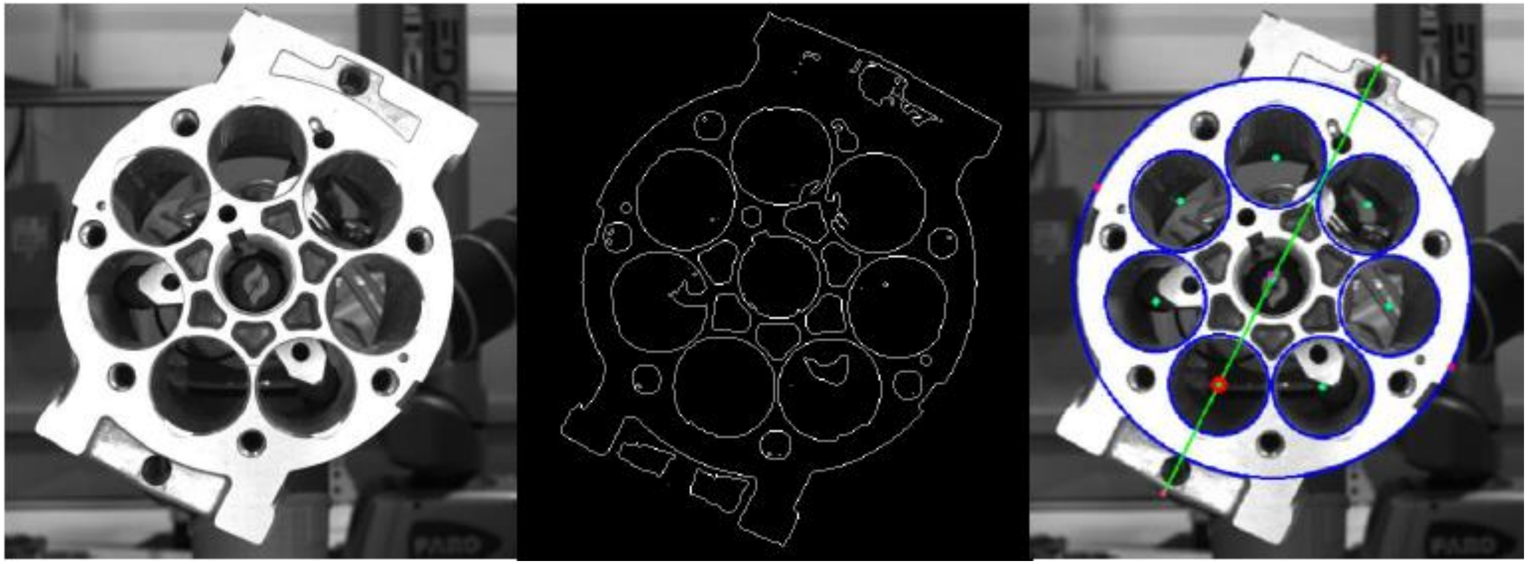
- Open Computer Vision Library (OpenCv) - http://opencv.org/
  - Focused on 2D images
  - 2D Image processing
  - Video
  - Sensor calibration
  - 2D features
  - GUI
  - GPU acceleration



http://opencv.org

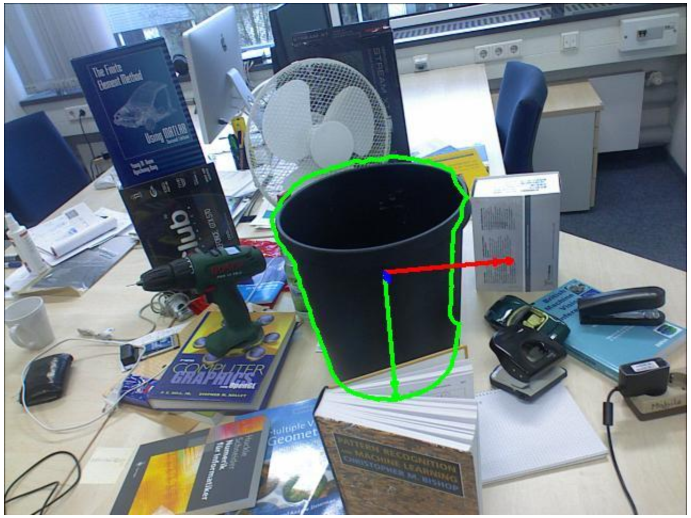
OpenCV tutorial

- Perform image processing to determine pump orientation (roll angle)
- Github tutorial link
- Training Wiki link



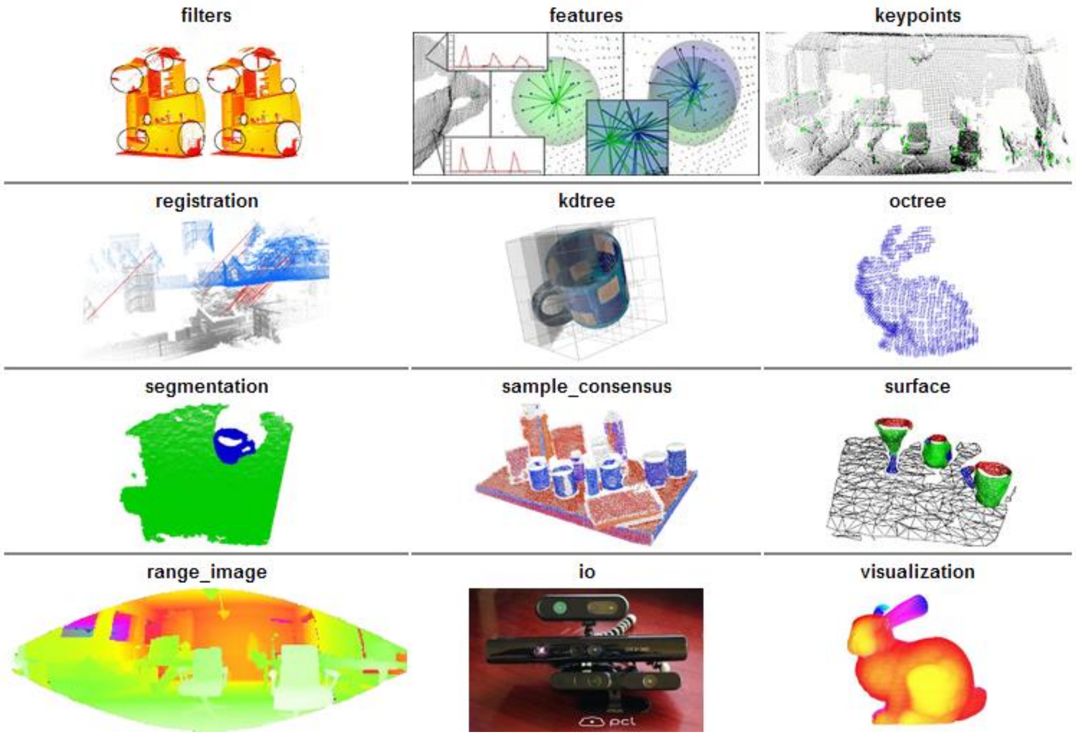
Perception Libraries (OpenCV)

- Open CV 3.2
  - Has more 3D tools
    - LineMod
      - https://www.youtube.com/watch?v=vsThfxzIUjs
    - PPF
  - Has opencv_contrib
    - Community contributed code
    - Some tutorials



Perception Libraries (PCL)

- Point Cloud Library (PCL) - http://pointclouds.org/
  - Focused on 3D Range(Colorized) data



http://pointclouds.org

Perception Libraries (PCL)

- PCL Command Line Tools
  - sudo apt install pcl-tools
  - Tools (140+)
    - pcl_viewer
    - pcl_point_cloud_editor
    - pcl_voxel_grid
    - pcl_sac_segmentation_plane
    - pcl_cluster_extraction
    - pcl_passthrough_filter
    - pcl_marching_cubes_reconstruction
    - pcl_normal_estimation
    - pcl_outlier_removal

ROS Bridges

- OpenCV & PCL are external libraries
- "Bridges" are created to adapt the libraries to the ROS architecture
  - OpenCV: https://index.ros.org/p/vision_opencv/
  - PCL: https://index.ros.org/p/pcl_ros/
    - Standard Nodes (PCL Filters): http://ros.org/wiki/pcl_ros#ROS_nodelets

Many More Libraries

- Many more libraries in the ROS Ecosystem
  - April Tag Detector
    https://github.com/ros-perception/ar_track_alvar/tree/ros2
  - AR Tracker
    https://github.com/christianrauch/apriltag_ros
  - ArUco Tag Detector/ Tracker
    https://aruco-ros2.readthedocs.io/en/latest/
  - Robot Self Filter (ROS1)
    http://www.ros.org/wiki/robot_self_filter

Exercise 4.1

- Play with PointCloud data
  - Play a point cloud file to simulate data coming from a Asus 3D sensor.
  - Matches scene for demo_manipulation
  - 3D Data in ROS 2
  - Use PCL Command Line Tools
- https://industrial-training-master.readthedocs.io/en/humble/_source/session4/2-Introduction-to-Perception.html

Review/Q&A

Session 3

ROS-Industrial

- Architecture
- Capabilities

Motion Planning

- Examine MoveIt Planning Environment
- Setup New Robot
- Motion Planning (Rviz)
- Motion Planning (C++)

Session 4

Moveit! Planning

Intro to Planners

Perception

- Calibration
- PointCloud File
- OpenCV
- PCL
- PCL Command Line Tools