ROS-Industrial Basic Developer's Training Class

March 2022

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

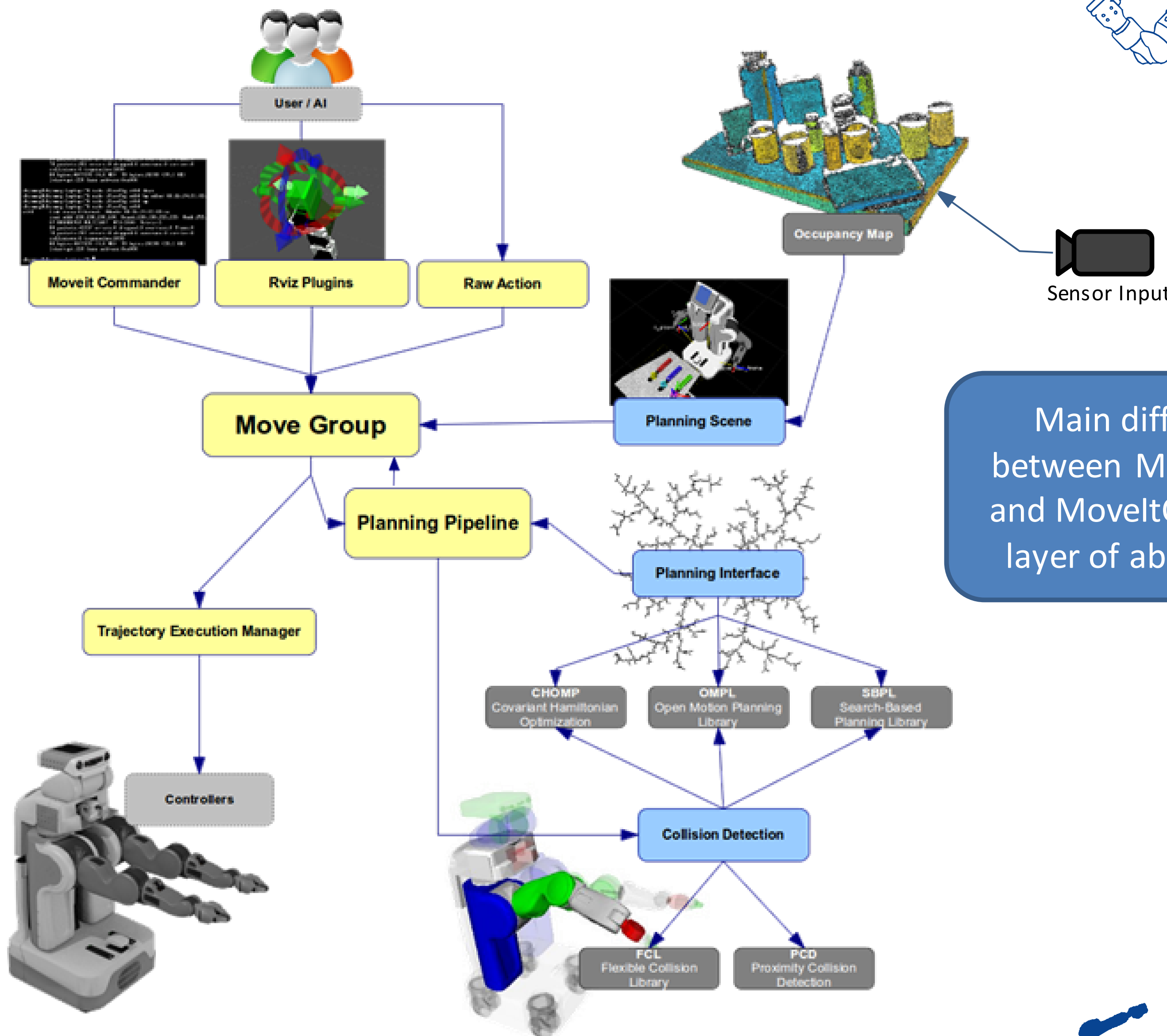
```
#include <moveit/moveit_cpp/moveit_cpp.h>
...
moveit_cpp::MoveItCpp::Ptr moveItCpp = make_shared(node);
moveit_cpp::PlanningComponent::Ptr planner = make_shared(“arm", moveItCpp);

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:

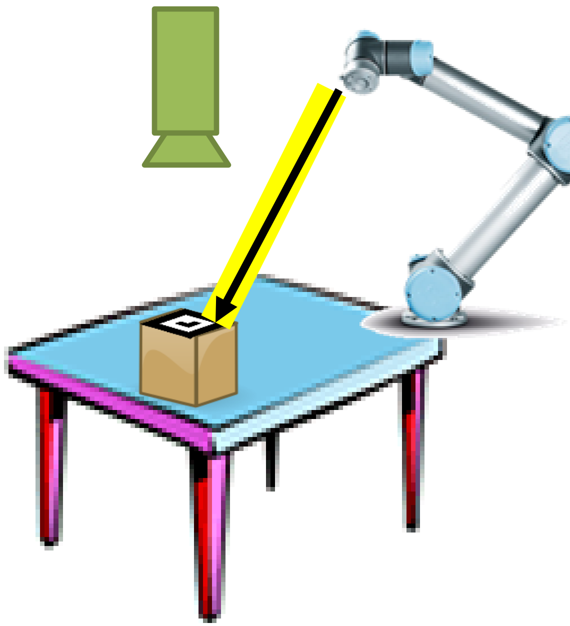
```
robot_state::RobotState joints.setStateValues(names, positions);
planner.setGoal(joints);
```

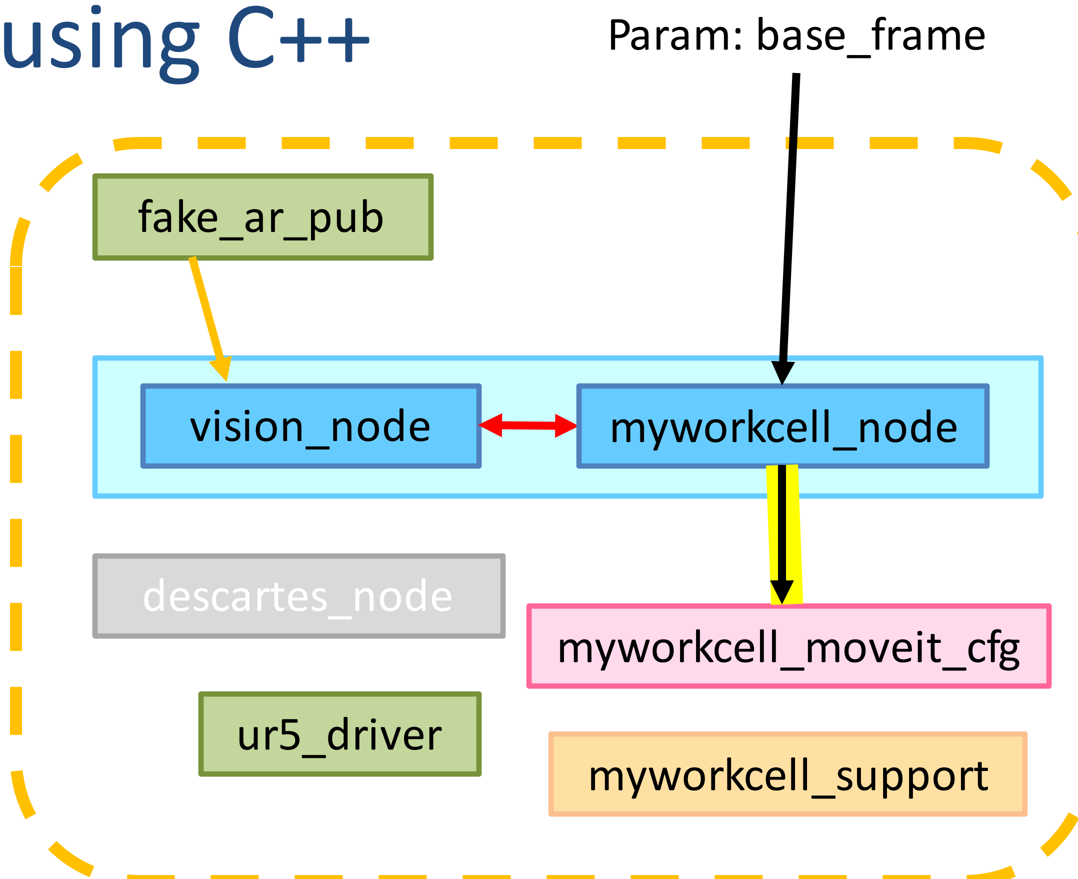
Cartesian position:

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

Exercise 4.0:

Motion Planning using C++

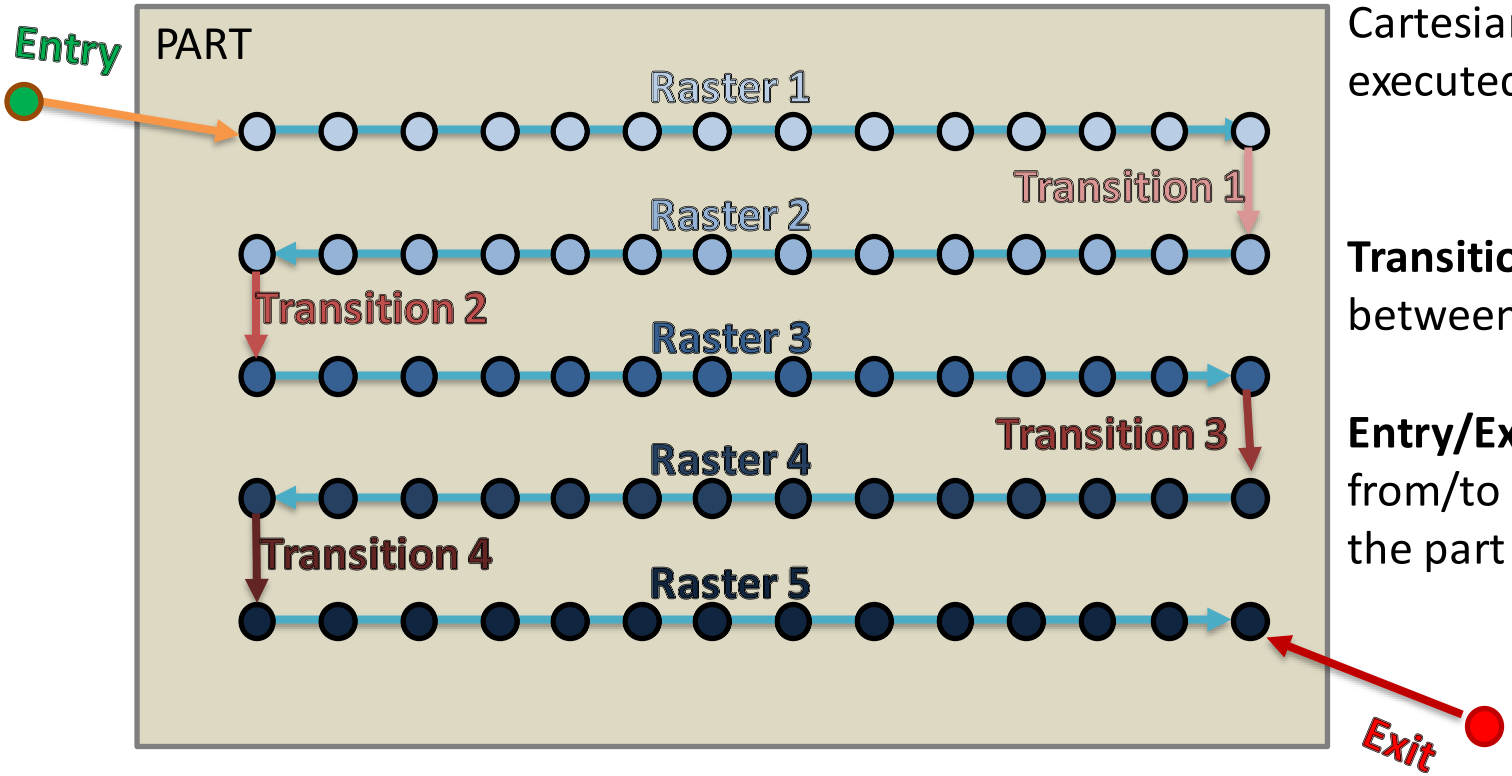




Intro to Planners

- Basic Toolpath Plan
- Planning Workflows
- Common Motion Planners
  - Descartes
  - TrajOpt
  - OMPL
  - Iterative Spline Parameterization
- Motion Planning Environments
- Planning Pipeline/Task Constructors

Toolpath Plan Example



Definitions

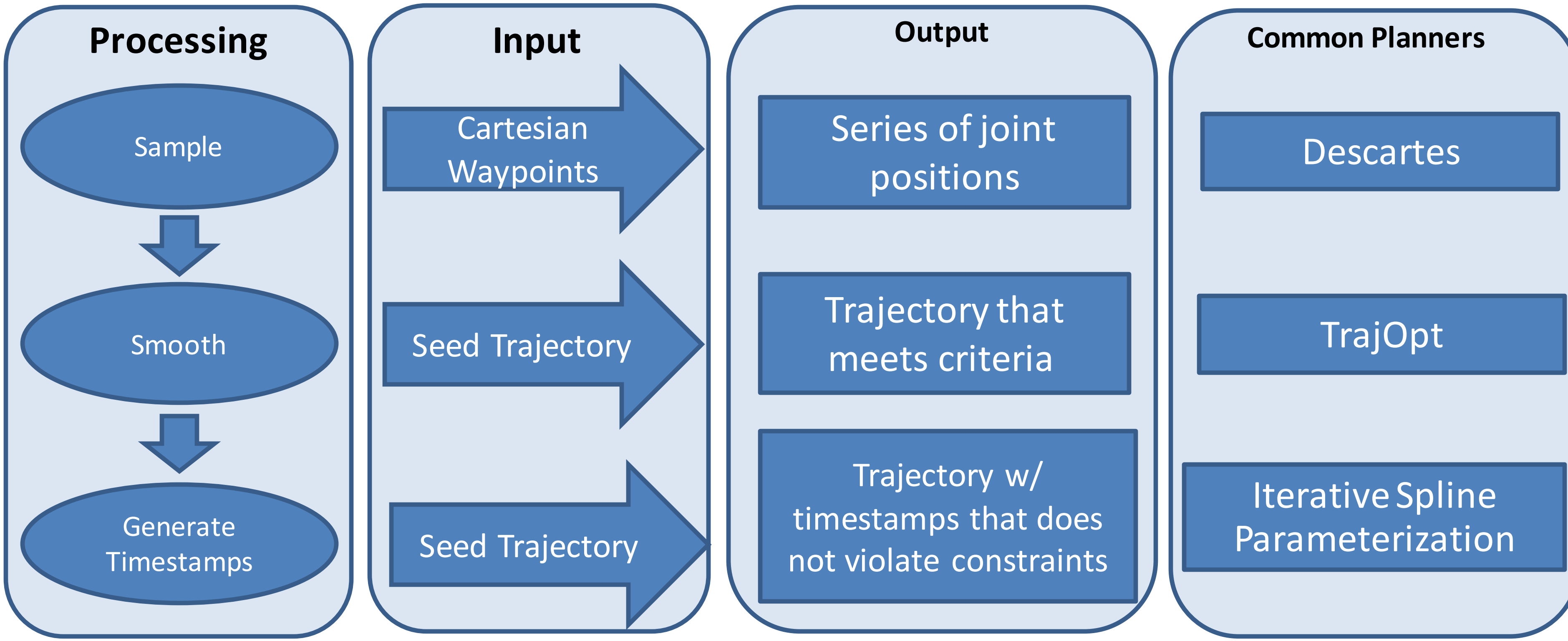
Raster - A series of specified Cartesian waypoints to be executed without breaking*

Transition - A freespace move between rasters

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

*depends on applic

Processing Planners



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

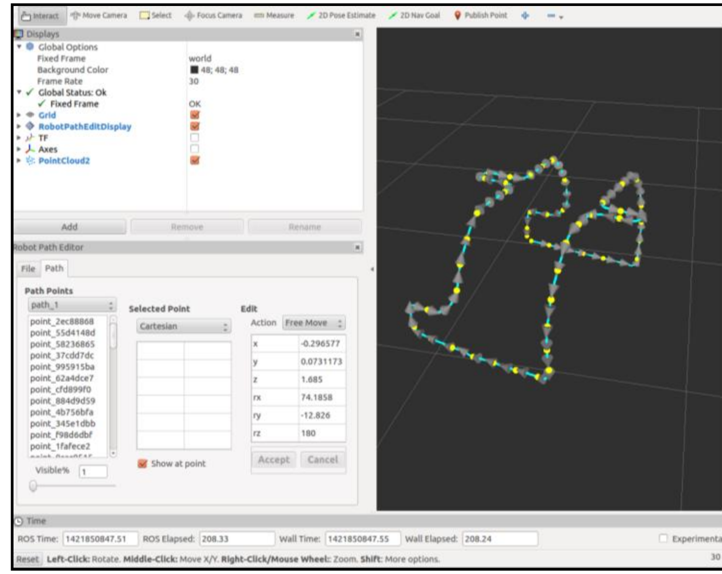
- TrajOpt/OMPL
- TrajOpt
- Iterative Spline Parameterization

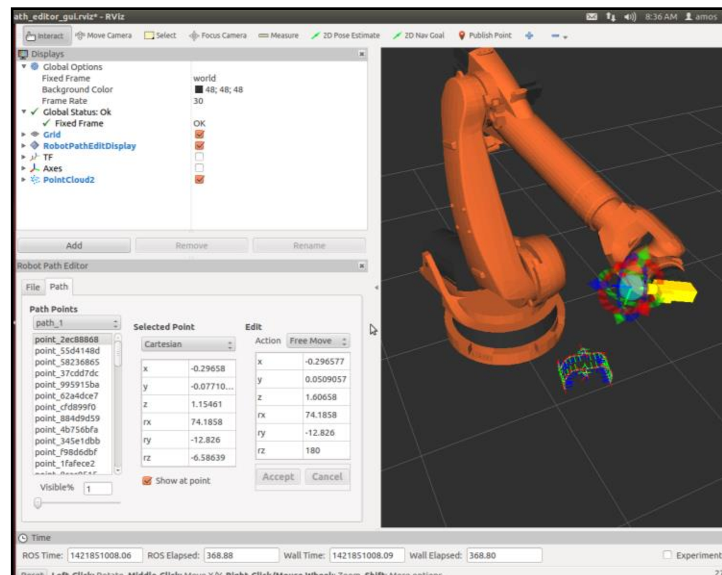
Descartes

In short:

Globally optimum; sampling-based search; Captures “tolerances”

- Inverse kinematics solver (waypoint -> joint state)
- Waypoint sampler
  - Fixed -> n solutions per waypoint, generally 8
  - Axial -> (n)*(360°/ sample angle)
  - Extra axis (7 DOF/rail/gantry)-> n*(extra axis1 samples)*(extra axis2 samples)*…
- Vertex evaluator
  - Account for certain configurations that may be in violation (DCS on Fanuc)
- Edge evaluator
  - Account for joint flips
- Environment collision checker
  - Specify allowed collision distance



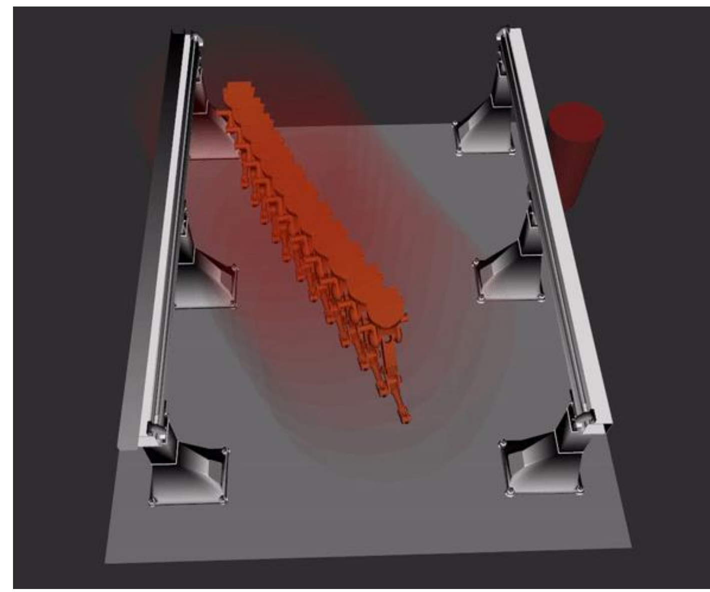


TrajOpt

In short:

Optimize existing trajectory on constraints (distance from collision, joint limits, etc.)

- All parameters have a coefficient that can be increased/decreased to change it's influence
- Collision parameters (cost or constraint)
  - Use weighted sum – combines collisions to be a single term
  - Safety margin – how far of collision distance must be maintained
  - Safety margin buffer – distance beyond safety margin to still use in optimization
- Smoothing (cost)
  - Velocity
  - Acceleration
  - Jerk
- Joint/Cartesian (cost or constraint) – Set a specified joint or cartesian DOF to be more or less valued
  - Example: Set the 6th term in the Cartesian coefficient to be 0 to allow rotation about the z axis
- Avoid singularities (cost)
- Longest valid segment – Resolution to check validity of state (as opposed to just checking discretely at each point in the seed)
- Other user defined parameters (cost or constraint)



OMPL

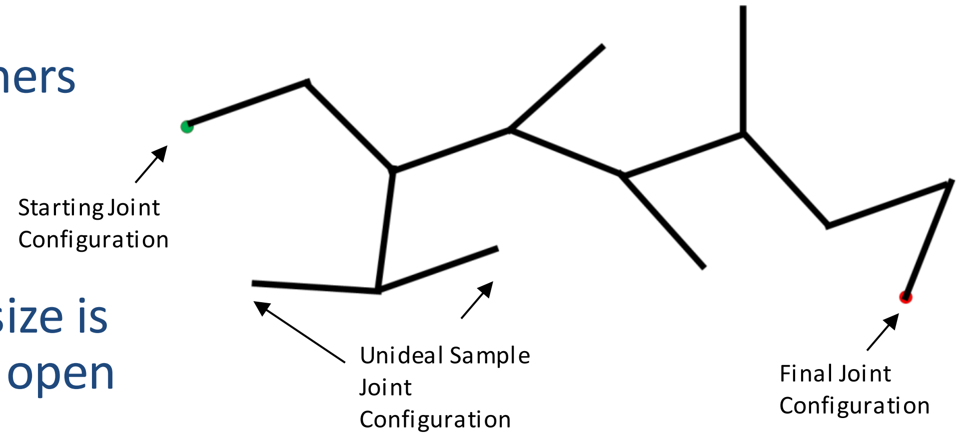
Open-source Motion Planning Library

In short:

Stochastic sampling; is a family of planners

Planners we often use:

- RRT
  - Parameters
    - Range: how long each step size is
      - Longer range solves big open spaces faster
      - Smaller range helps get through tight spots
    - Goal bias: How frequently the algorithm tries to move to the goal
- RRT-Connect (most commonly used by MRTD)
  - 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



Iterative Spline Parameterization

- **Input:**
  - Trajectory of joint states at a fine mesh of waypoints (position & velocity)
- **Output:**
  - **Reduces** the **velocity** & *interpolated* **acceleration** between these waypoints **to be within specified bounds**/ make for smoother motions, and **adjusts the timestamps** as necessary to follow the same cartesian path as before
- **Assumptions:**
  - *No corners cut by low-level controllers/ will travel though every point*
- **Parameters:**
  - Joint max velocities
  - Joint max accelerations

Motion Planning Environments

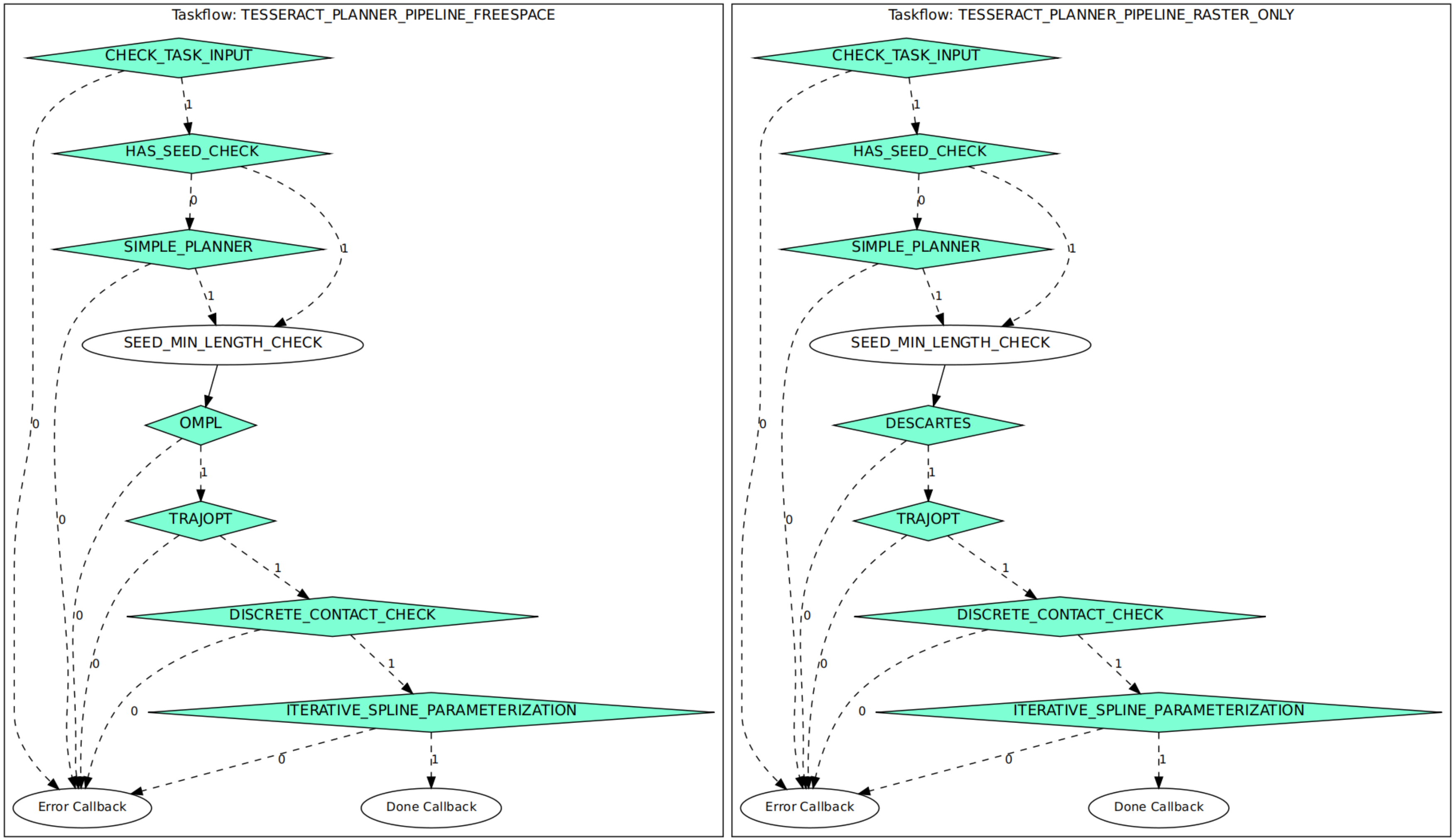
Interfaces used to generate motion plans can be:

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

Examples we use:

- MoveIt!/MoveIt!2
  - Easy to use, wizard & UI interfaces, broad toolset
- Tesseract
  - Enables very complex planning, different toolset

Planning Pipeline/Task Construction



INTRODUCTION TO PERCEPTION

Outline

- Camera Calibration
- 3D Data Introduction
  - Exercise 4.2
- Explanation of the Perception Tools Available in ROS
- Intro to PCL tools
  - Exercise 4.2

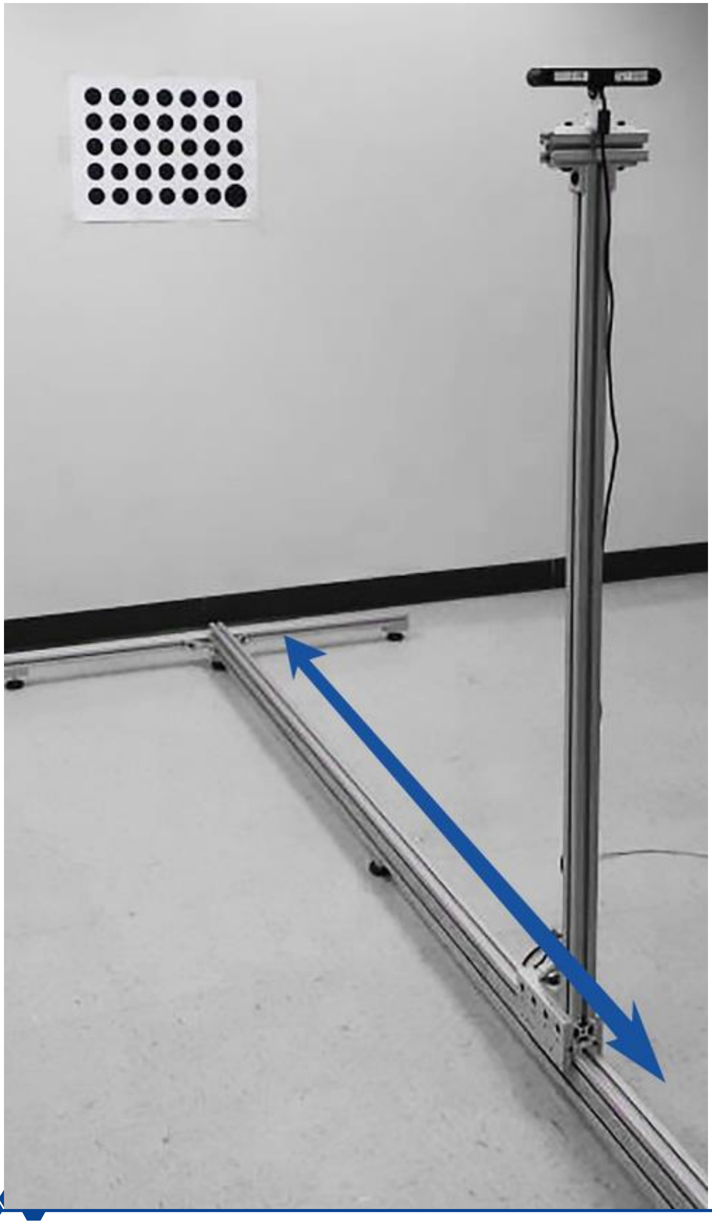
Objectives

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

Industrial Calibration

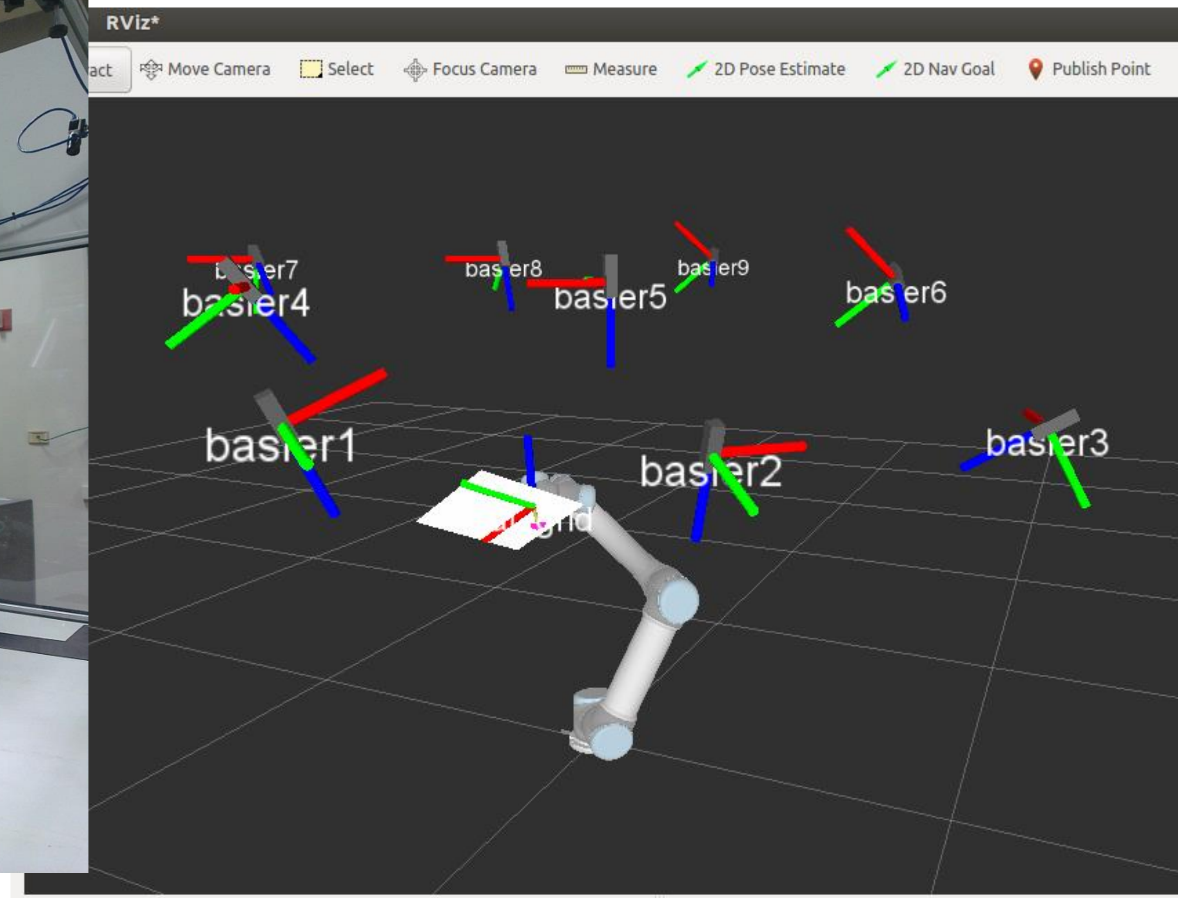
- Perform intrinsic and extrinsic calibration
- Continuously improving library
- Resources, library
  - Github link
  - Wiki link
- Resources, tutorials
  - Github industrial calibration tutorials link
  - Training Wiki link

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



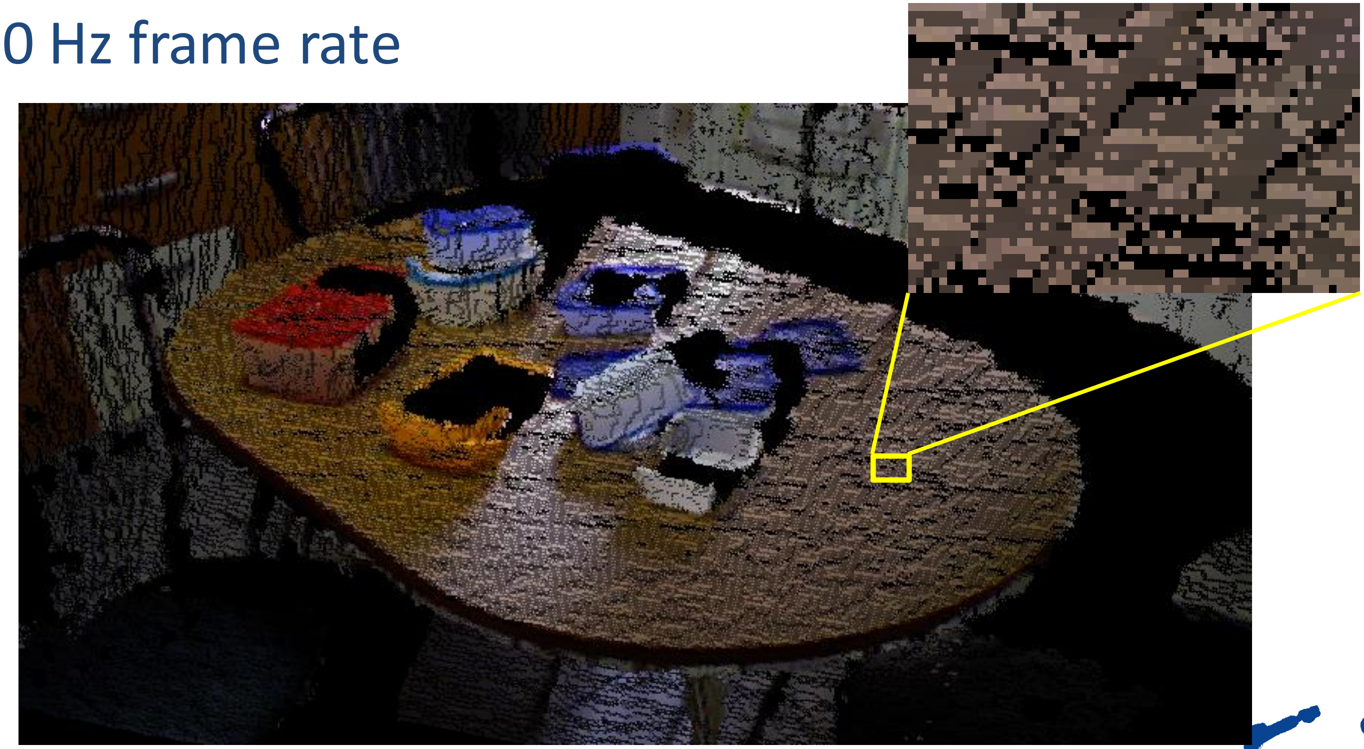
https://www.youtube.com/watch?v=MJFtEr_Y4ak

3D Cameras

- RGBD cameras, TOF cameras, stereo vision, 3D laser scanner
- Driver for Asus Xtion camera and the Kinect (1.0) is in the package openni2_launch
- Driver for Kinect 2.0 is in package iai_kinect2 (github link)
- https://rosindustrial.org/3d-camera-survey

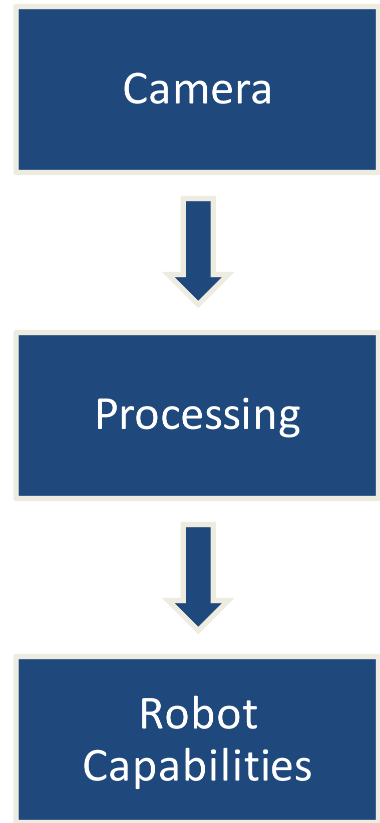
3D Cameras

- 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



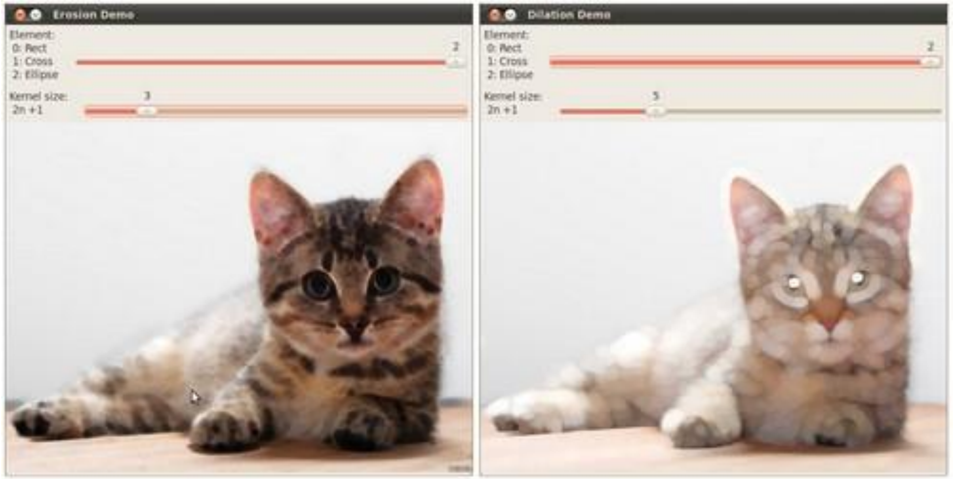
Perception Tools

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

- Focus on PCL tools for exercise

Perception Libraries (OpenCV)

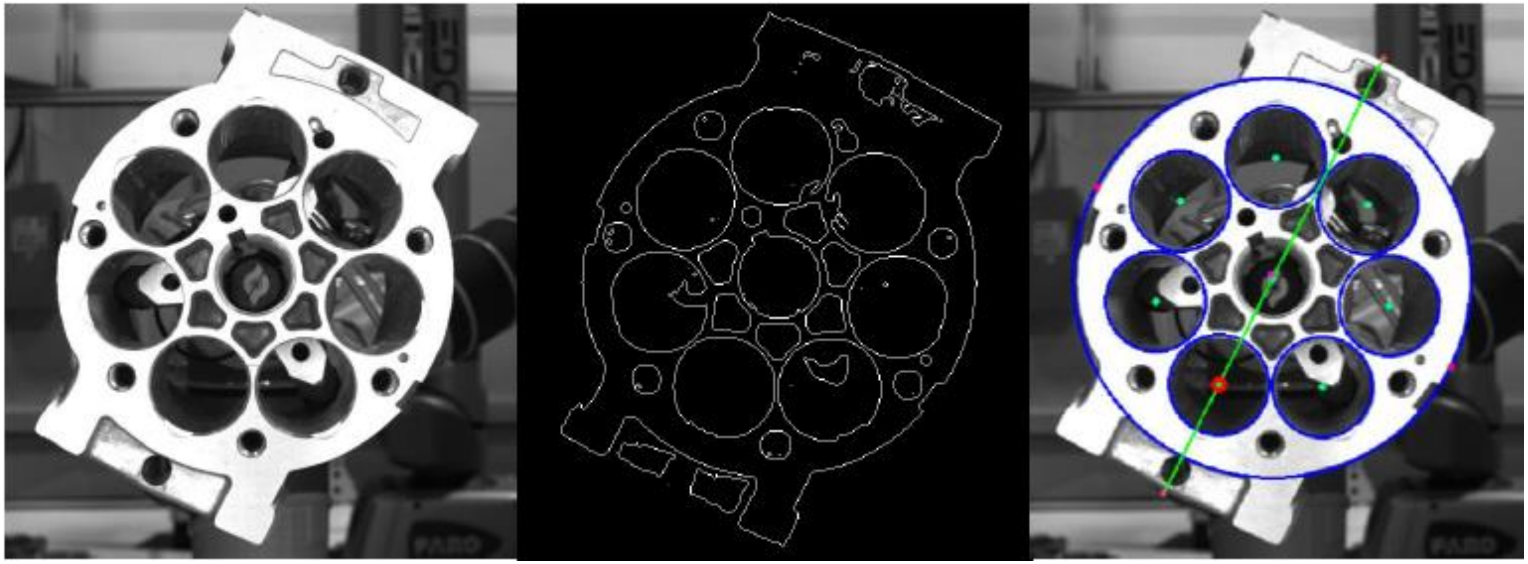
- Open Computer Vision Library (OpenCv) - http://opencv.org/
  - Focused on 2D images
  - 2D Image processing
  - Video
  - Sensor calibration
  - 2D features
  - GUI
  - GPU acceleration



http://opencv.org

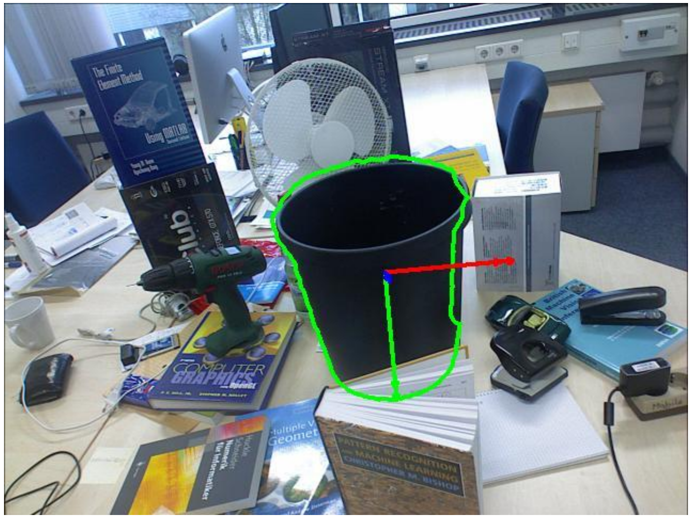
OpenCV tutorial

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



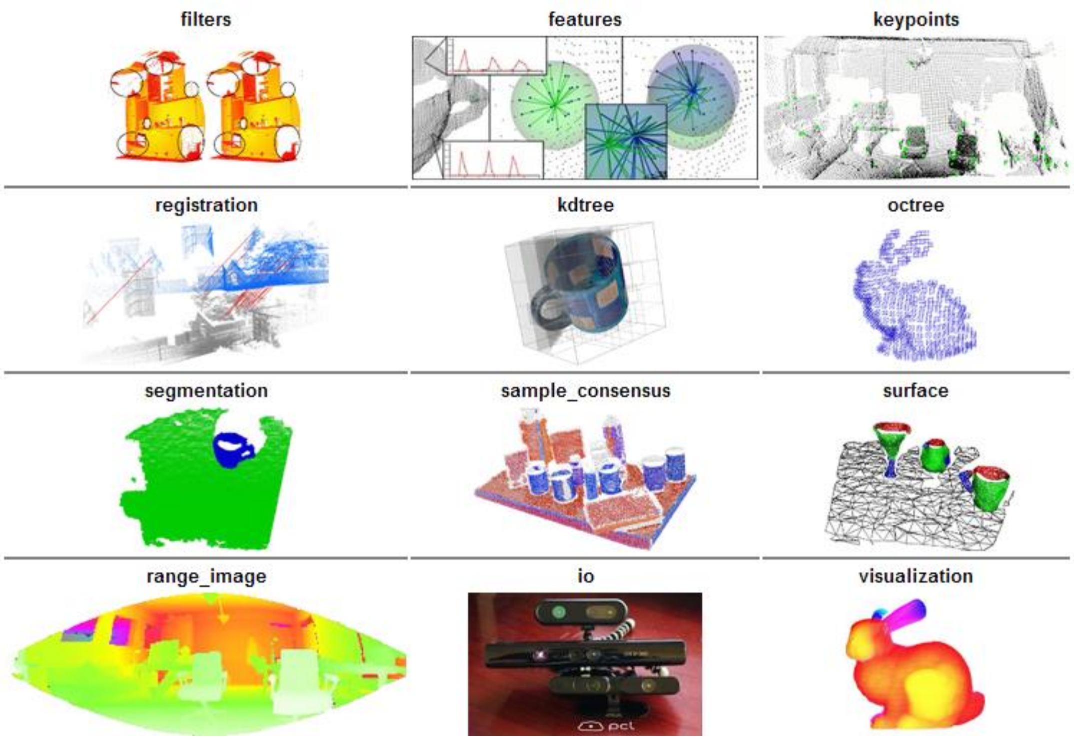
Perception Libraries (OpenCV)

- 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: http://ros.org/wiki/vision_opencv
  - PCL: http://ros.org/wiki/pcl_ros
    - Standard Nodes (PCL Filters): http://ros.org/wiki/pcl_ros#ROS_nodelets

Many More Libraries

- Many more libraries in the ROS Ecosystem
  - AR Tracker
    http://www.ros.org/wiki/ar_track_alvar
  - Robot Self Filter
    http://www.ros.org/wiki/robot_self_filter

Exercise 4.2

- 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://github.com/ros-industrial/industrial_training/wiki/Introduction-to-Perception

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