ROS-Industrial Basic Developer's Training Class

June 2017

Southwest Research Institute

Session 4:

More Advanced Topics (Descartes and Perception)

Southwest Research Institute

Motion Planning in C++

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

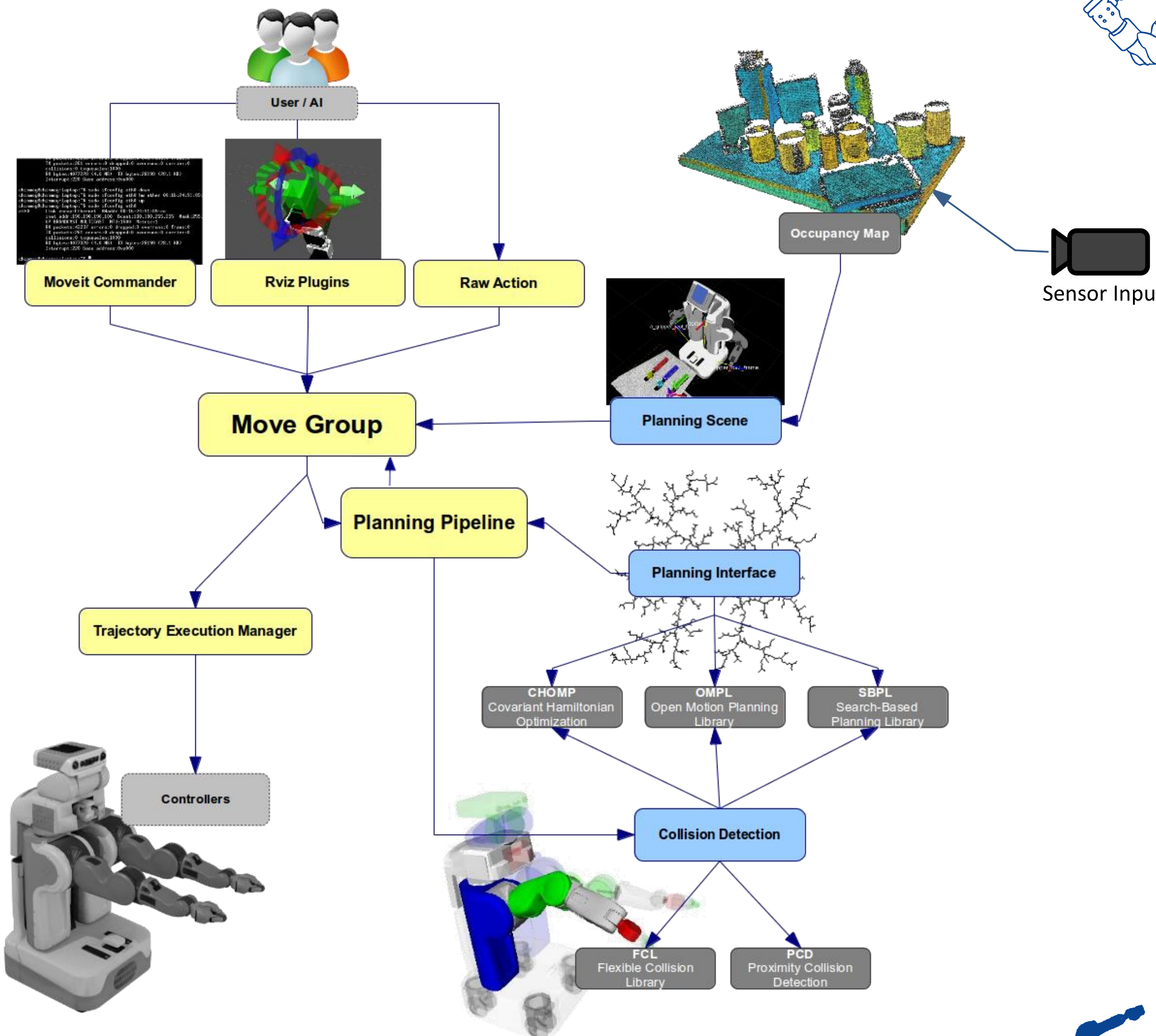
move_group_interface

```
#include <moveit/move_group_interface/move_group_interface.h>
...
Moveit::planning_interface::MoveGroupInterface group("manipulator");
group.setRandomTarget();
group.move();
```

3 lines = collision-aware path planning & execution

Amazing!

Reminder: MoveIt! Complexity



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

Motion Planning in C++

Pre-defined position:

```
group.setNamedTarget("home");
group.move();
```

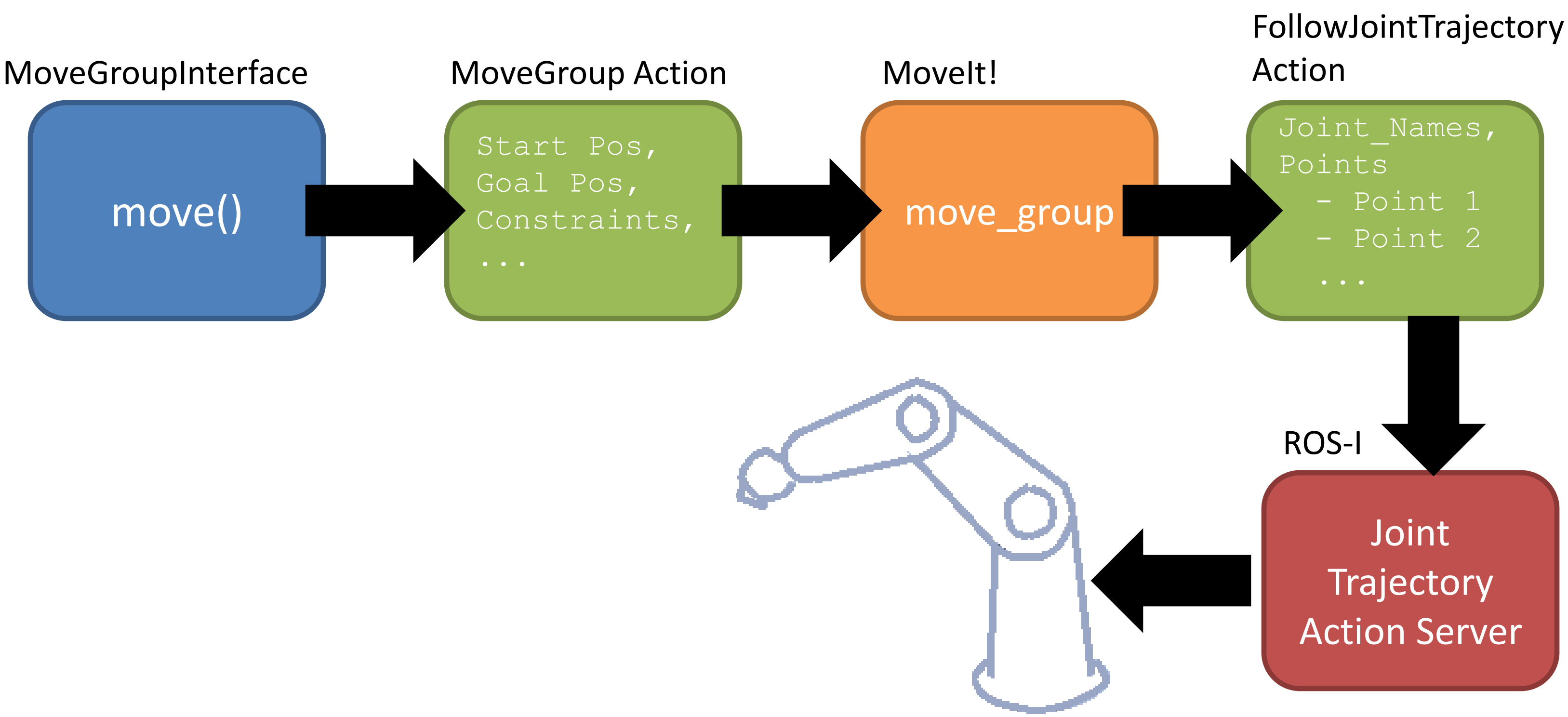
Joint position:

```
map<string, double> joints = my_function();
group.setJointValueTarget(joints);
group.move();
```

Cartesian position:

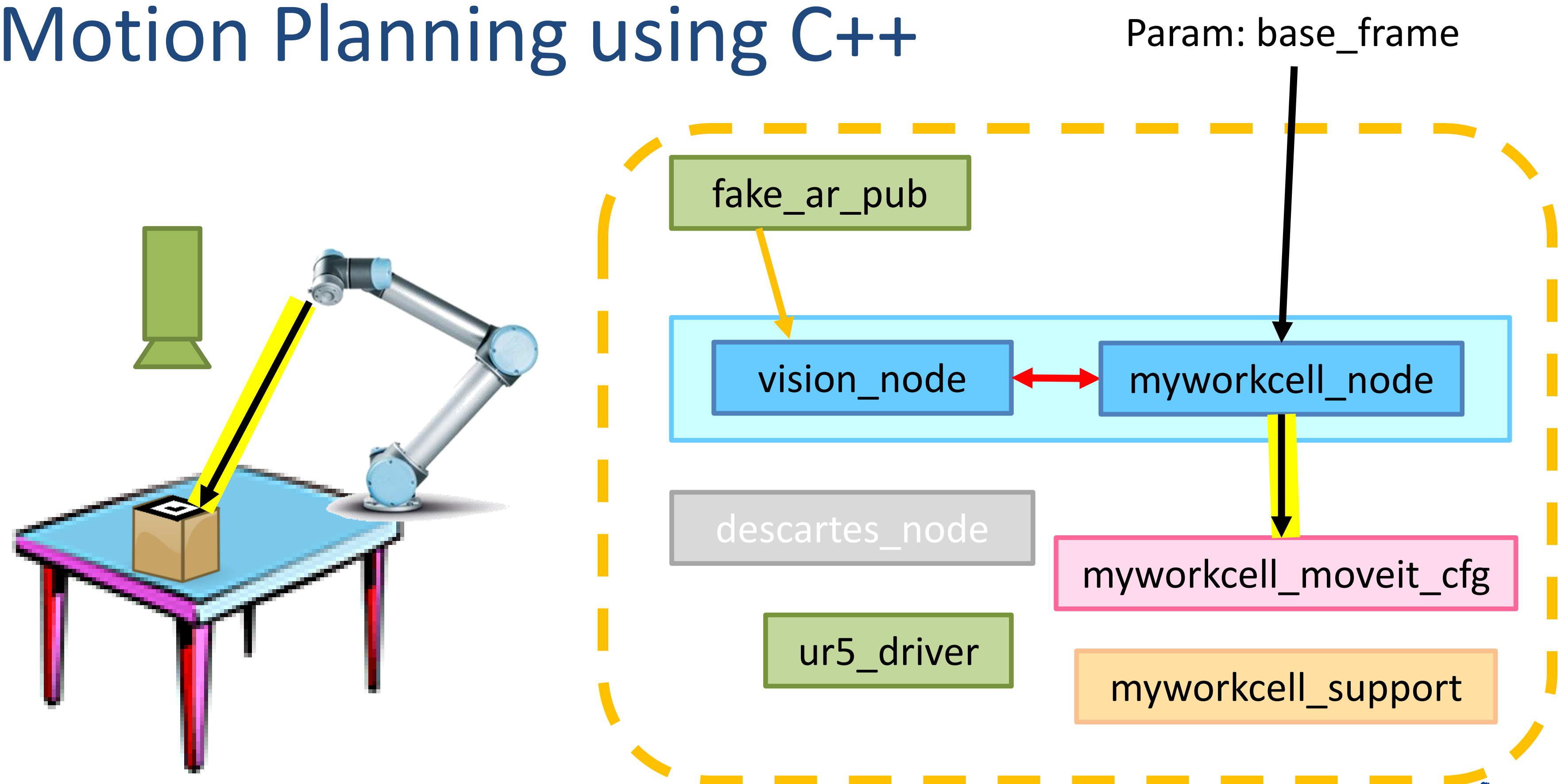
```
Affine3d pose = my_function();
group.setPoseTarget(joints);
group.move();
```

Behind the Scenes



Exercise 4.0:

Motion Planning using C++



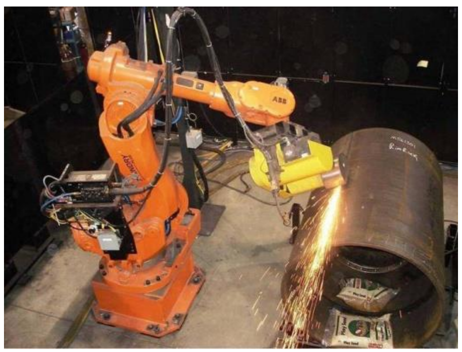
INTRODUCTION TO DESCARTES

Outline

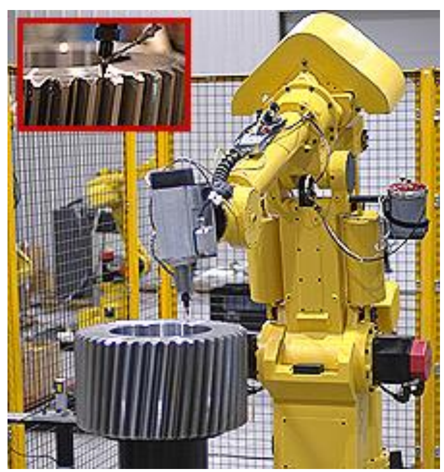
- Introduction
- Overview
  - Descartes architecture
- Path Planning
  - Exercise 4.1

Introduction

- Application Need:
  - Semi-constrained trajectories: traj. DOF < robot DOF



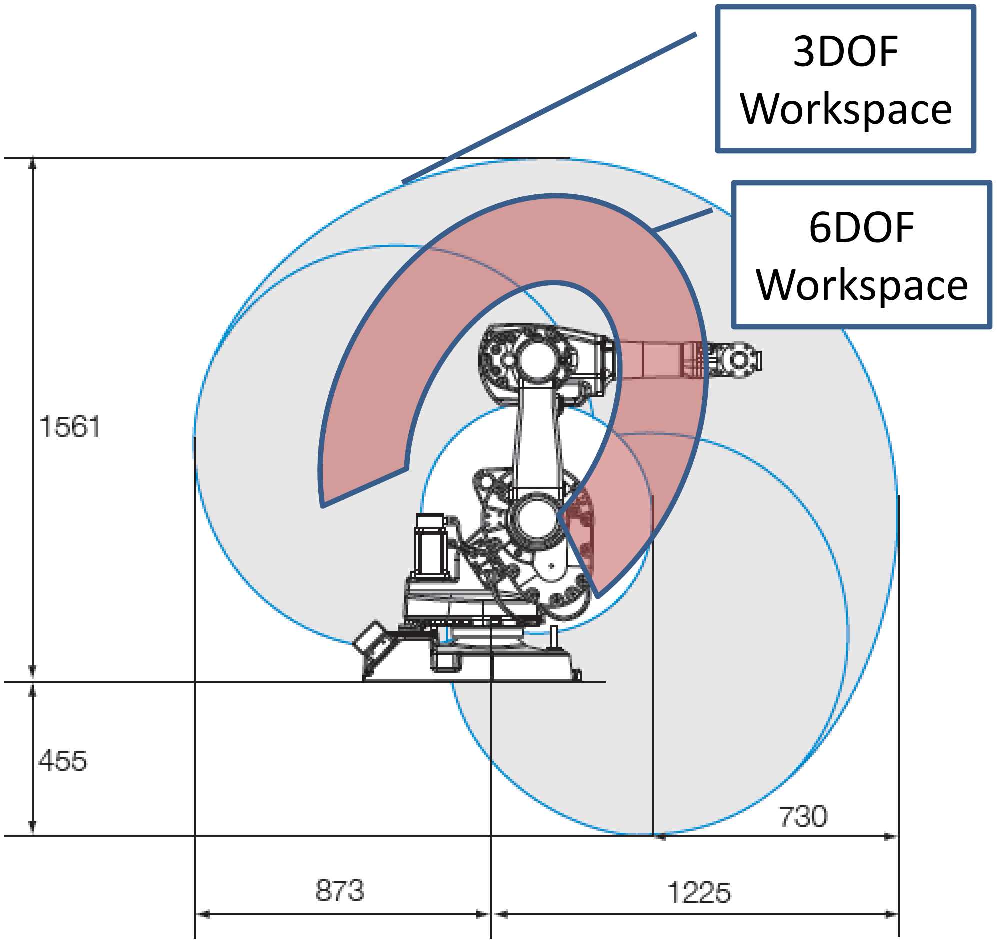


Current Solution

- Arbitrary assignment of 6DOF poses, redundant axes -> IK
- Limited guarantee on trajectory timing
- Limitations
  - Reduced workspace
  - Relies on human intuition
  - Collisions, singularities,

joint limits

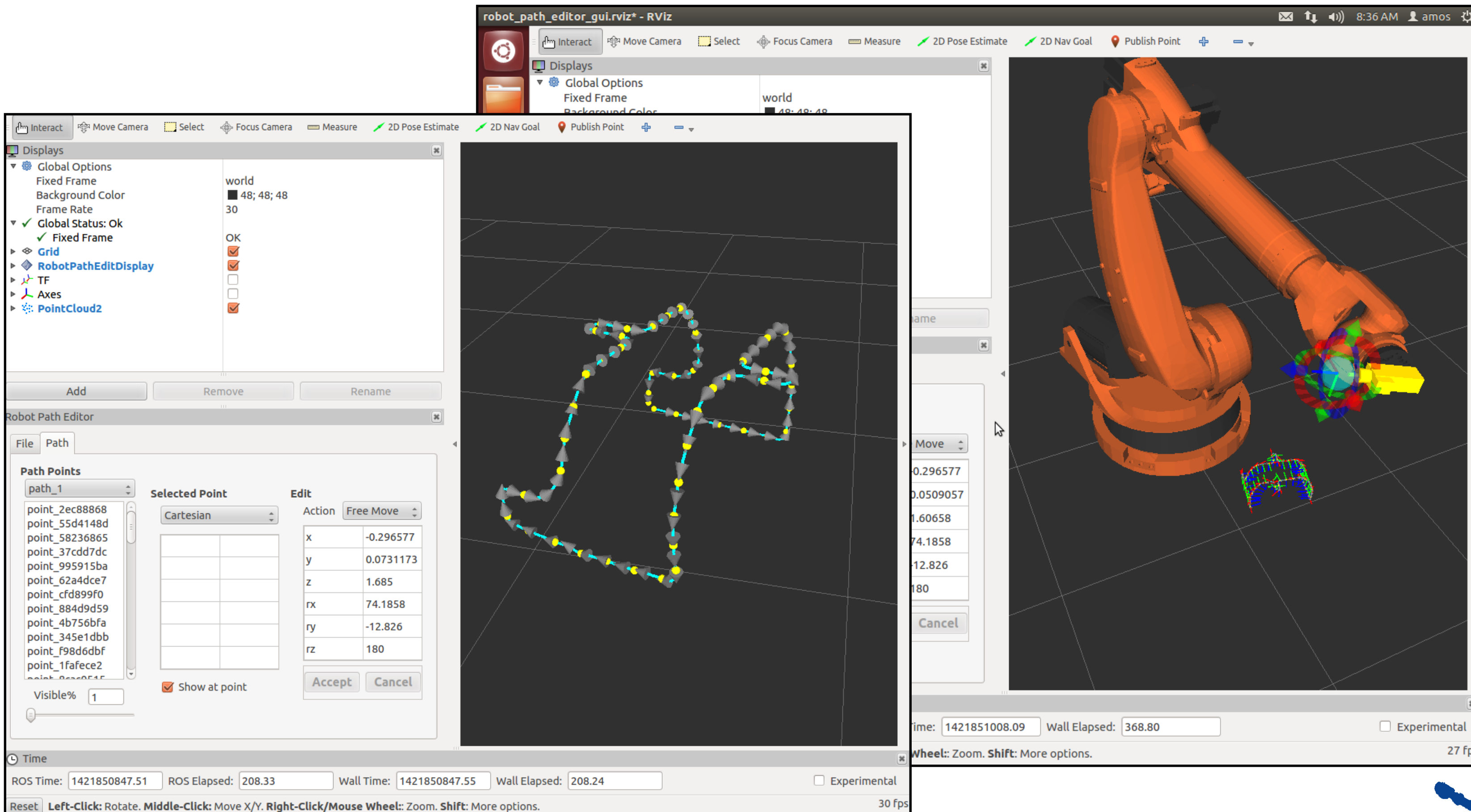


Descartes

- Planning library for semi-constrained trajectories
- Requirements
  – Generate well behaved plans that minimize joint motions
  – Find easy solutions fast, hard solutions with time
  – Handle hybrid trajectories (joint, Cartesian, specialized points)
  – Fast re-planning/cached planning

Descartes Use Case

- Robotic Routing



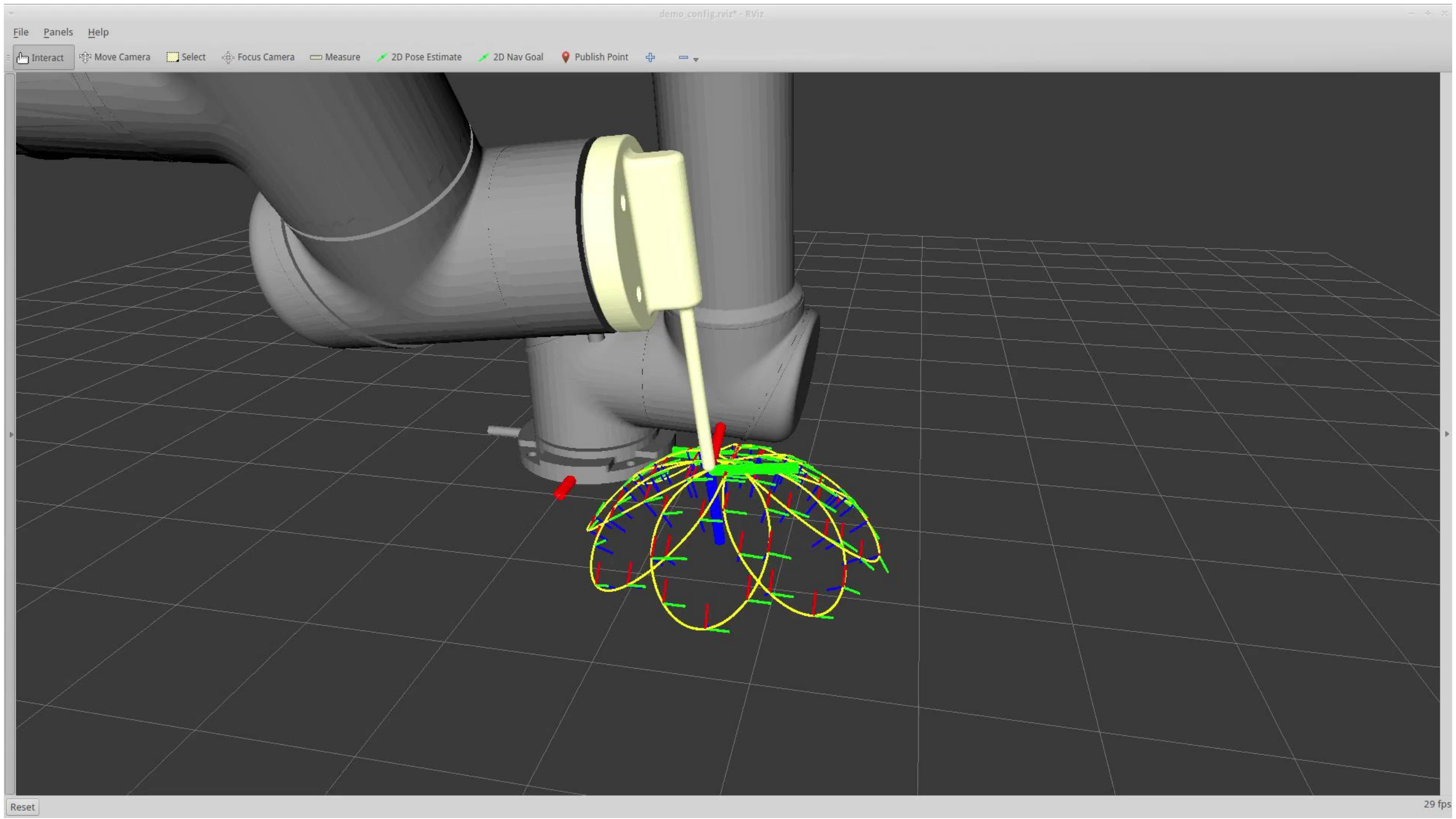
Other Uses

- Robotic Blending

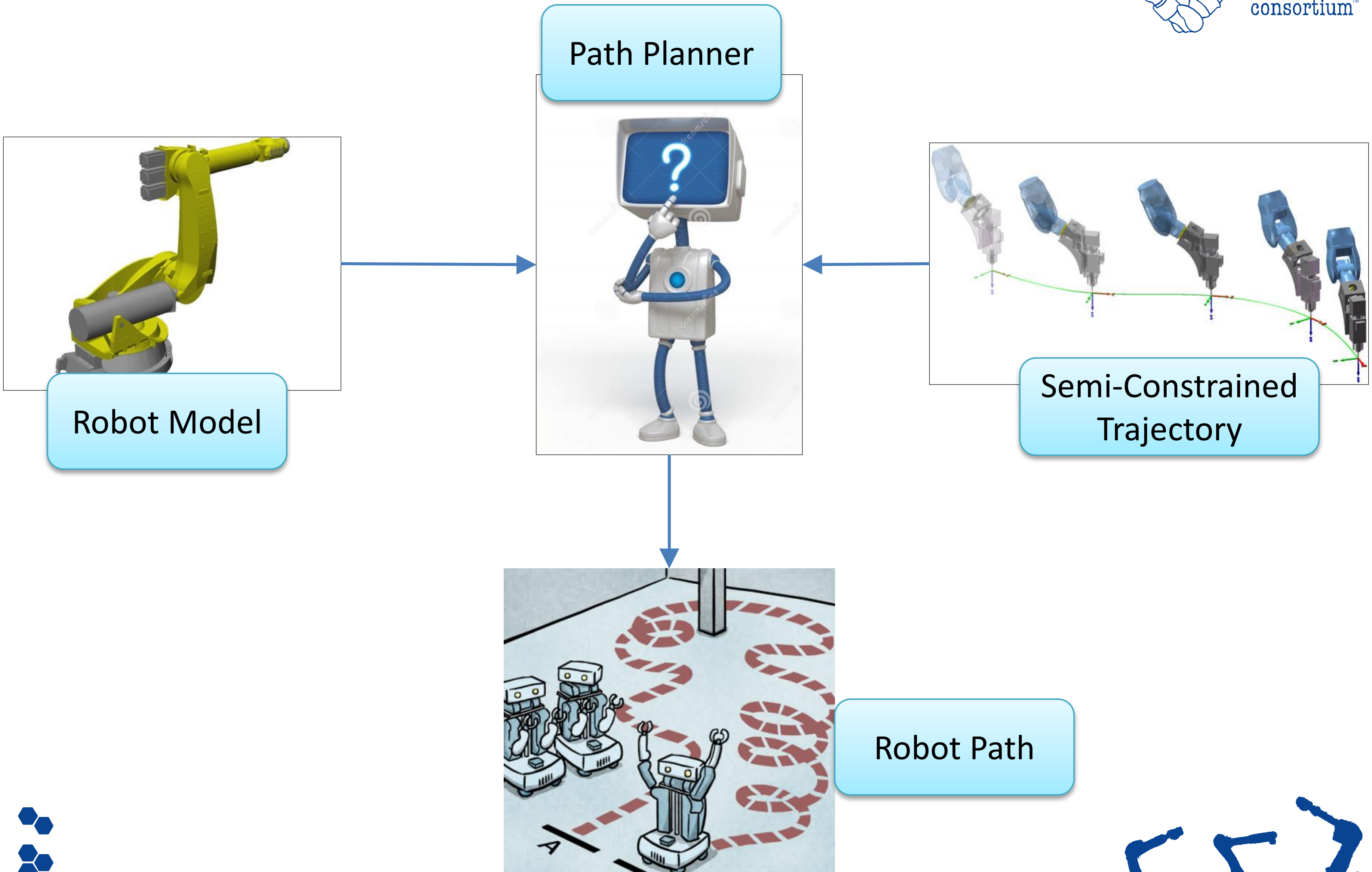
Open Source Details

- Public development: https://github.com/ros-industrial-consortium/descartes
- Wiki Page: http://wiki.ros.org/descartes
- Acknowledgements:
  - Dev team: Dan Solomon (former SwRI), Shaun Edwards (former SwRI), Jorge Nicho (SwRI), Jonathan Meyer (SwRI), Purser Sturgeon(SwRI)
  - Supported by: NIST (70NANB14H226), ROS-Industrial Consortium FTP

Descartes Demonstration



Descartes Architecture



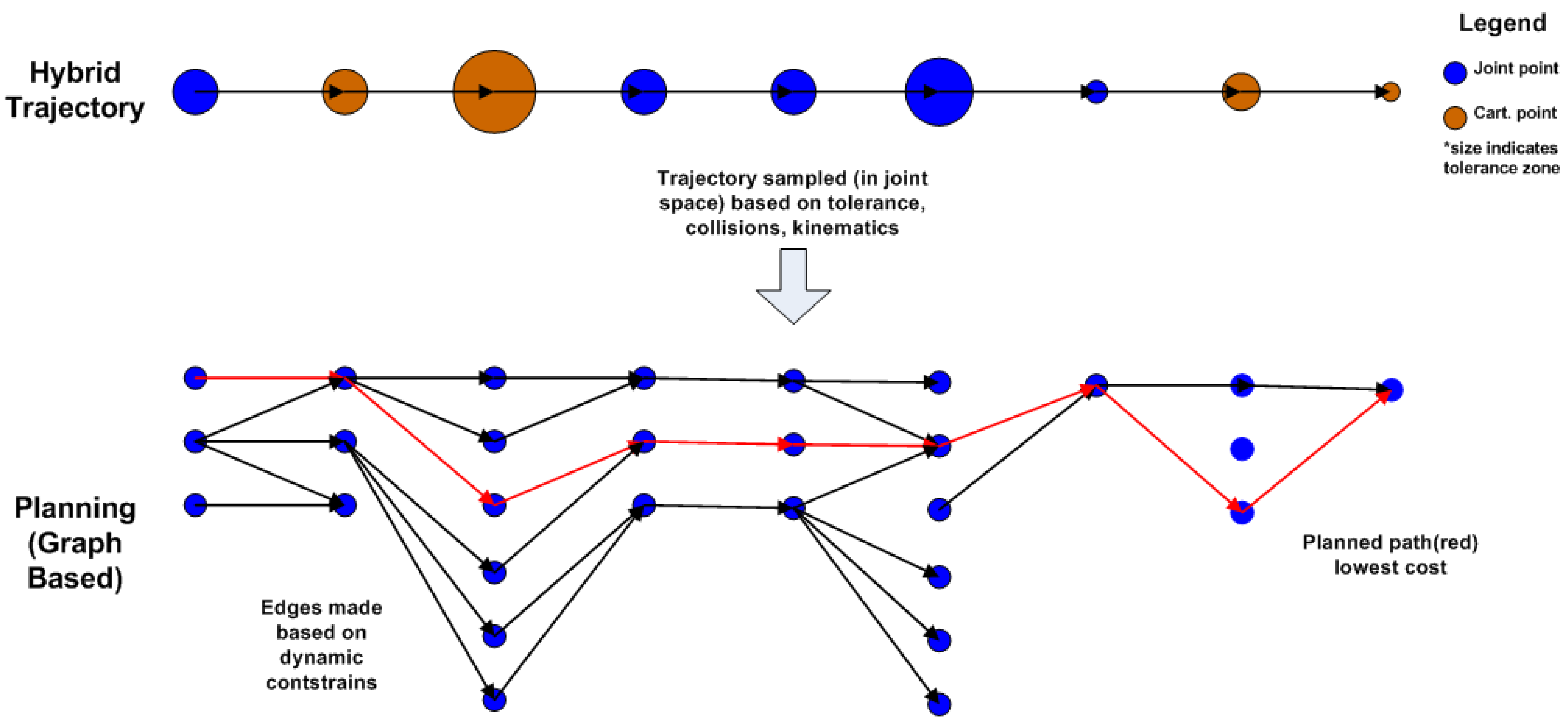
Descartes Interfaces

- Trajectory Point
  - Robot independent
  - Tolerance (fuzzy)
  - Timing
- Robot Model
  - IK/FK
  - Validity (Collision checking, limits)
  - Similar to MoveIt::RobotState, but with **getAllIK**
- Planner
  - Trajectory solving
  - Plan caching/re-planning

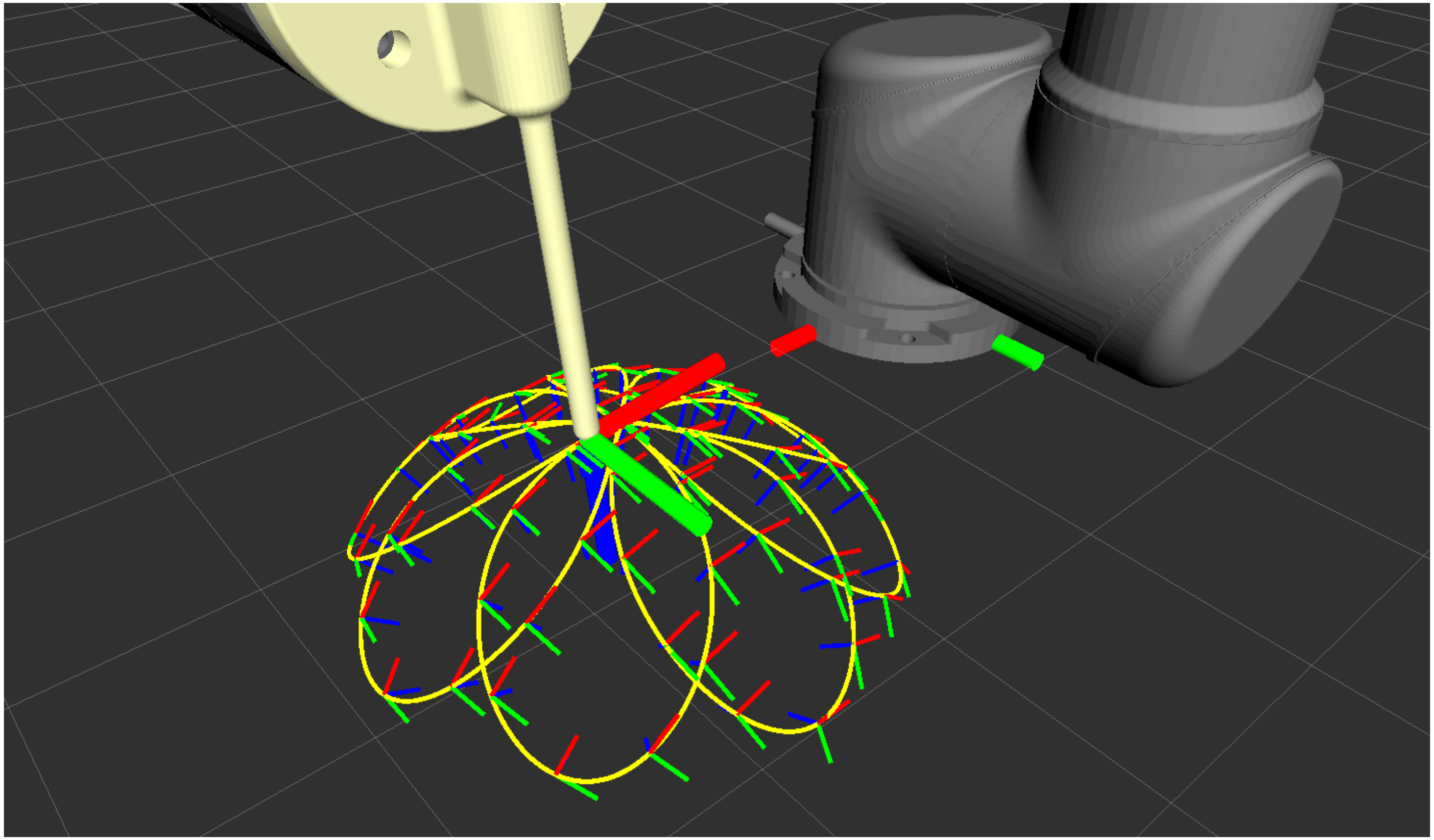
Descartes Implementations

- Trajectory Points
  - Cartesian point
  - Joint point
  - AxialSymmetric point (5DOF)
- Robot Model
  - MoveIt wrapper (working with MoveIt to make better)
  - FastIK wrappers
  - Custom solution
- Planners
  - Dense – graph based search
  - Sparse – hybrid graph based/interpolated search

Descartes Implementations



DESCARTES IMPLEMENTATIONS



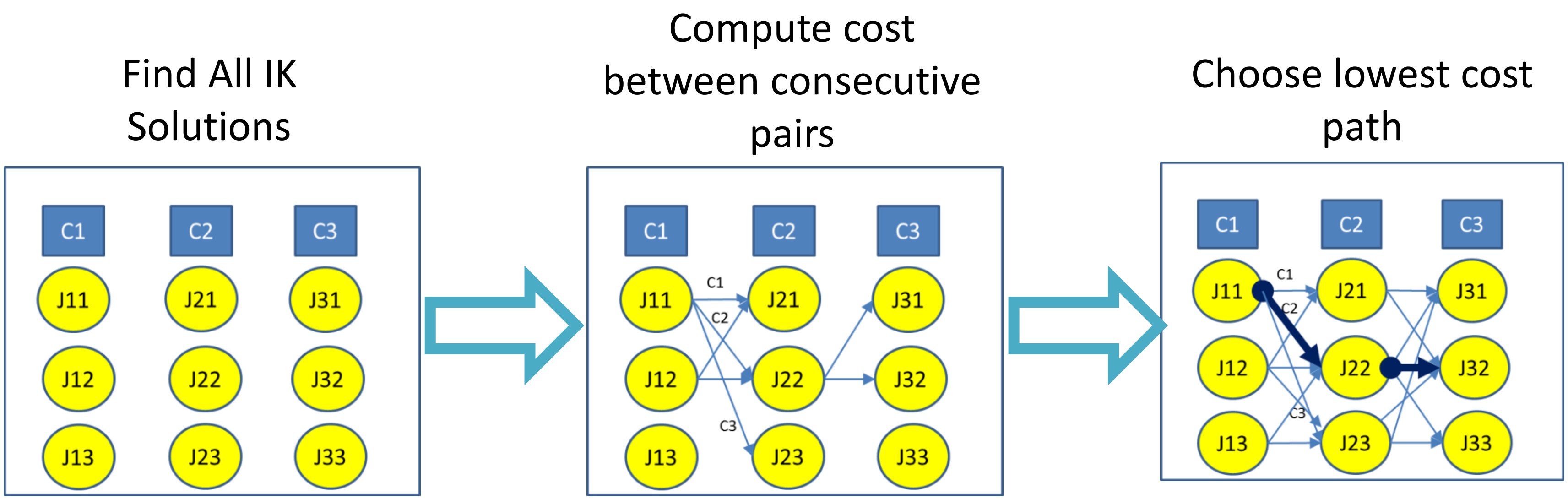
You specify these “points”, and Descartes finds shortest path through them.

Descartes Path Planning

- Planners
  - Planners are the highest level component of the Descartes architecture.
  - Take a trajectory of points and return a valid path expressed in joint positions for each point in the tool path.
  - Two implementations
    - DensePlanner
    - SparsePlanner

Descartes Path Planning

- Dense Planner
  - Finds a path through the points that minimizes the joint motion.



Descartes Path Planning

- Dense Planner
  - Search graph uses joint solutions as vertices and the movement costs as edges
  - Applies Dijkstra’s algorithm to find the lowest cost path from a start to and end configuration.

Descartes Path Planning

- Create a trajectory of **AxialSymetricPt** points.
- Store all of the points in the **traj** array.

```
for( …)
 {
   …

   descartes_core::TrajectoryPtPtr  cart_point (
      new descartes_trajectory::AxialSymmetricPt (
                                        tool_pose ,
                                        1.0f,
                                        descartes_trajectory::AxialSymmetricPt::Z_AXIS));
   traj.push_back(cart_point);
 }
```

Descartes Path Planning

- Create and **initialize** a **DensePlanner**.
- Verify that initialization succeeded.

```
descartes_planner::DensePlanner planner;
 if (planner.initialize( robot_model_ptr ))
 {
              …
 }
```

Descartes Path Planning

- Use **planPath(…)** to plan a robot path.
- Invoke **getPath(…)** to get the robot **path** from the planner.

```
std::vector < descartes_core::TrajectoryPtPtr > path;
 if ( planner.planPath(  traj  ) )
 {
   if ( planner. getPath( path ) )
   {
               …
   }
               …
}
```

Descartes Path Planning

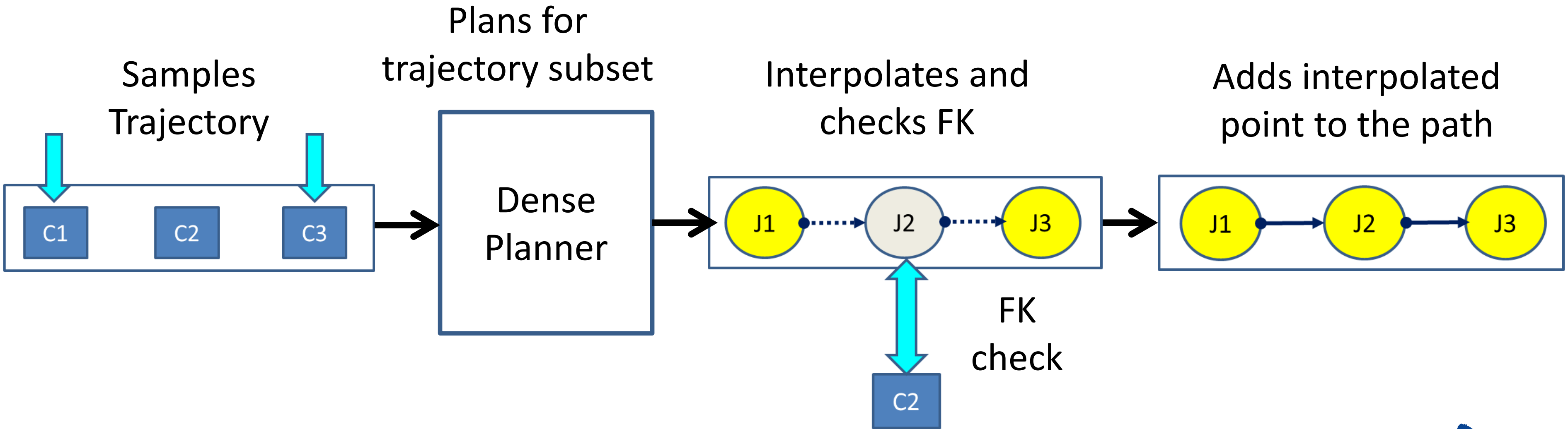
- Write a **for loop** to print all the joints poses in the planned path to the console.

```
std::vector< double >  seed ( robot_model_ptr->getDOF() );
for( … )
{
  std::vector <double> joints;
  descartes_core::TrajectoryPtPtr  joint_pt =  path[ i ];
  joint_pt -> getNominalJointPose (seed ,*robot_model_ptr , joints );

        // print joint values in joints
}
```

Descartes Path Planning

- Sparse Planner
  - Saves computational time by planning with a subset of the trajectory points and completing the path using joint interpolation.



Exercise 4.1

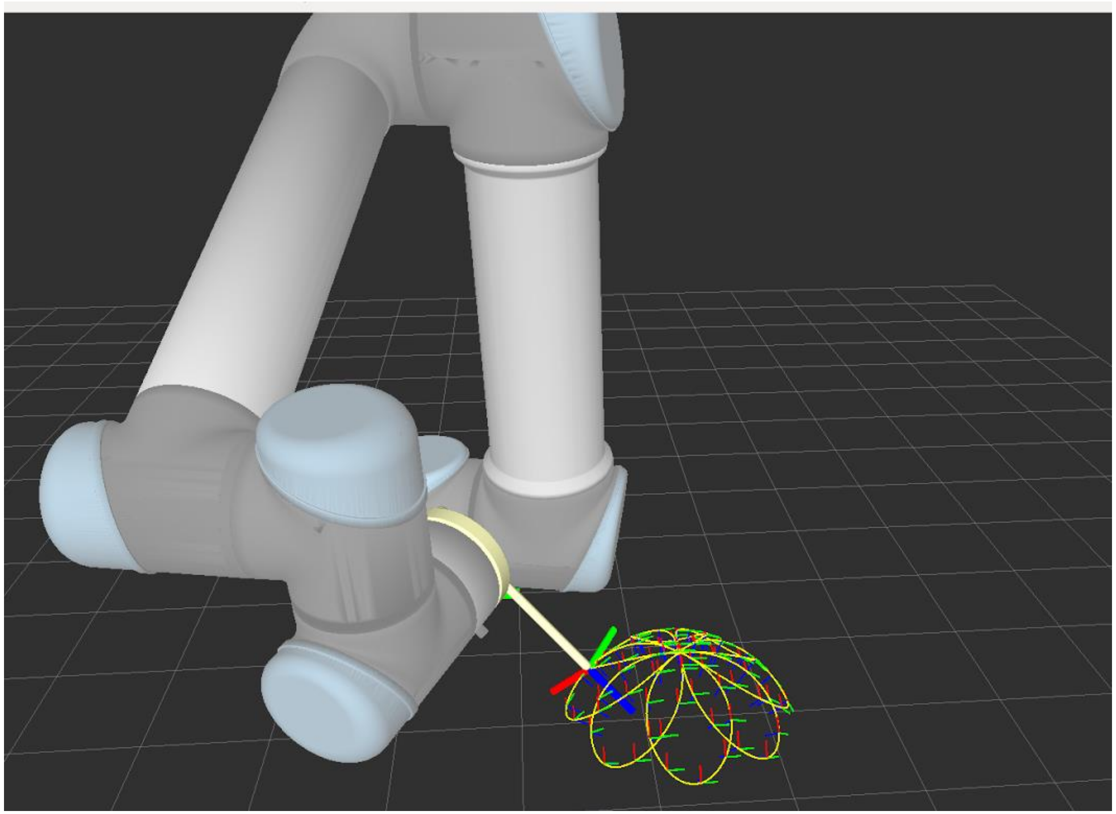
- Go back to the line where the **DensePlanner** was created and replace it with the **SparsePlanner**.
- Planning should be a lot faster now.

~~descartes_planner::DensePlanner planner~~;

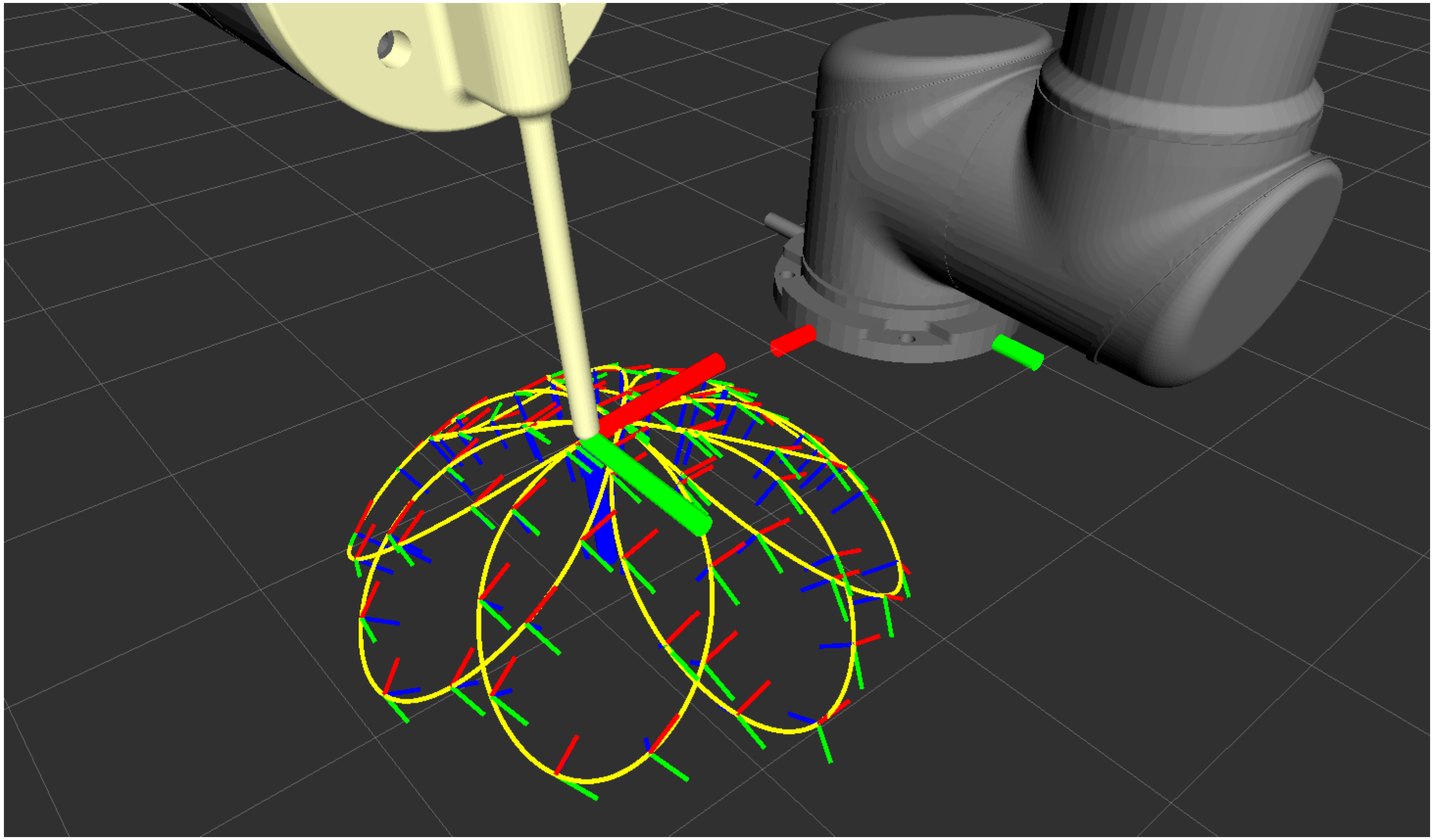
descartes_planner::SparsePlanner planner;

Exercise 4.1:

Descartes Path Planning



DESCARTES IMPLEMENTATIONS



These points have a free degree of freedom, but they don't have to.

Trajectory Point "Types"

- Trajectory Points
  - JointTrajectoryPt
    - Represents a robot joint pose. It can accept tolerances for each joint
  - CartTrajectoryPt
    - Defines the position and orientation of the tool relative to a world coordinate frame. It can also apply tolerances for the relevant variables that determine the tool pose.
  - AxialSymmetricPt
    - Extends the CartTrajectoryPt by specifying a free axis of rotation for the tool. Useful whenever the orientation about the tool's approach vector doesn't have to be defined.

Cartesian Trajectory Point

- Create a **CartTrajectoryPt** from a tool pose.
- Store the **CartTrajectoryPt** in a **TrajectoryPtPtr** type.

```
descartes_core::TrajectoryPtPtr  cart_point_ptr ( new
        descartes_trajectory::CartTrajectoryPt ( tool_pose ));
```

Axial Symmetric Point

- Use the **AxialSymmetricPt** to create a tool point with rotational freedom about z.
- Use **tool_pose** to set the nominal tool position.

```
descartes_core::TrajectoryPtPtr  free_z_rot_pt(
    new descartes_trajectory::AxialSymmetricPt(
        tool_pose,
        0.5f,
        descartes_trajectory::AxialSymmetricPt::Z_AXIS));
```

Joint Point

- Use the **JointTrajectoryPt** to "fix" the robot's position at any given point.
- Could be used to force a particular start or end configuration.

```
std::vector<double> joint_pose = {0, 0, 0, 0, 0, 0};
descartes_core::TrajectoryPtPtr   joint_pt(
    new descartes_trajectory::JointTrajectoryPt(joint_pose) );
```

Timing Constraints

- All trajectory points take an optional **TimingConstraint** that enables the planners to more optimally search the graph space.
- This defines the time, in seconds, to achieve this position from the previous point.

```
Descartes_core::TimingConstraint tm (1.0);
descartes_core::TrajectoryPtPtr  joint_pt(
    new descartes_trajectory::JointTrajectoryPt(joint_pose, tm) );
```

Robot Models

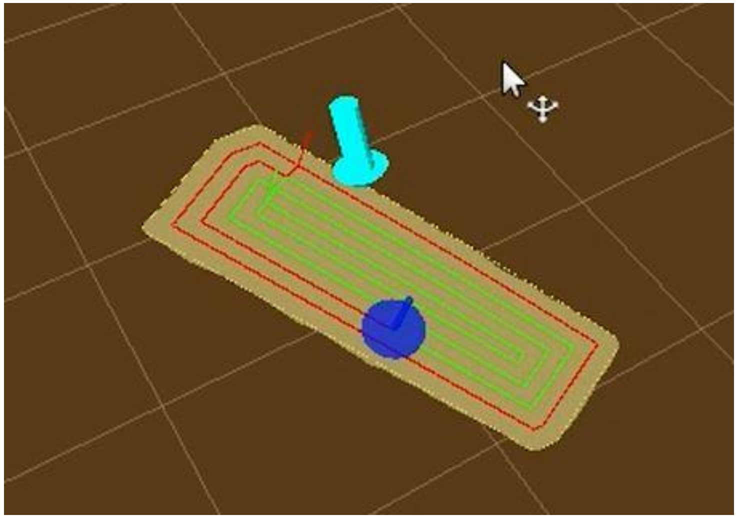
- Robot Model Implementations
  - **MoveitStateAdapter** :
    - Wraps moveit Robot State.
    - Can be used with most 6DOF robots.
    - Uses IK Numerical Solver.
  - **Custom Robot Model**
    - Specific to a particular robot (lab demo uses UR5 specific implementation).
    - Usually uses closed-form IK solution (a lot faster than numerical).
    - Planners solve a lot faster with a custom robot model.

Used in the Exercises

Used in the Lab

Descartes Input/Output

- Input:
  - Can come from CAD
  - From processed scan data
  - Elsewhere
- Output
  - Joint trajectories
  - Must convert to ROS format to work with other ROS components (see 4.0)



Common Motion Planners

Motion Planner	Application Space	Notes
Descartes	Cartesian path planning	Globally optimum; sampling-based search; Captures “tolerances”
CLIK	Cartesian path planning	Local optimization; Scales well with high DOF; Captures “tolerances”
STOMP	Free-space Planning	Optimization-based; Emphasizes smooth paths
OMPL / MoveIt	Free-space Planning	Stochastic sampling; Easy and convenient interface

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 4.3 (Now a Lab)

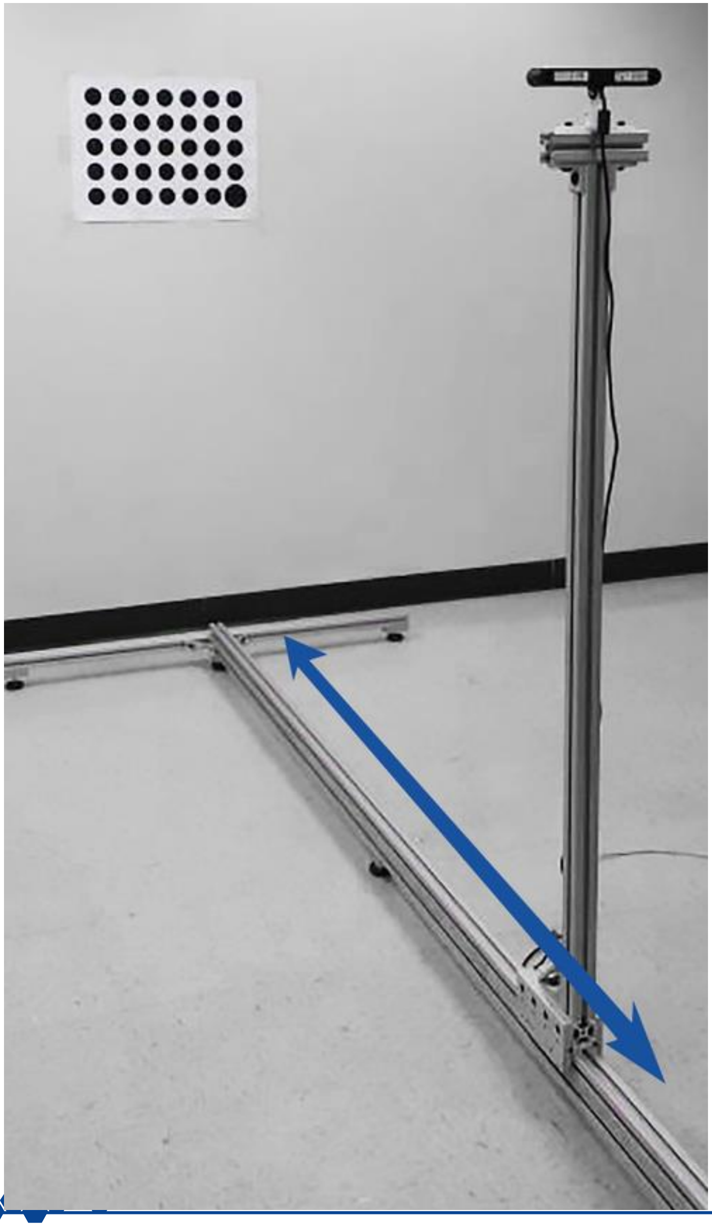
Objectives

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

Industrial Calibration

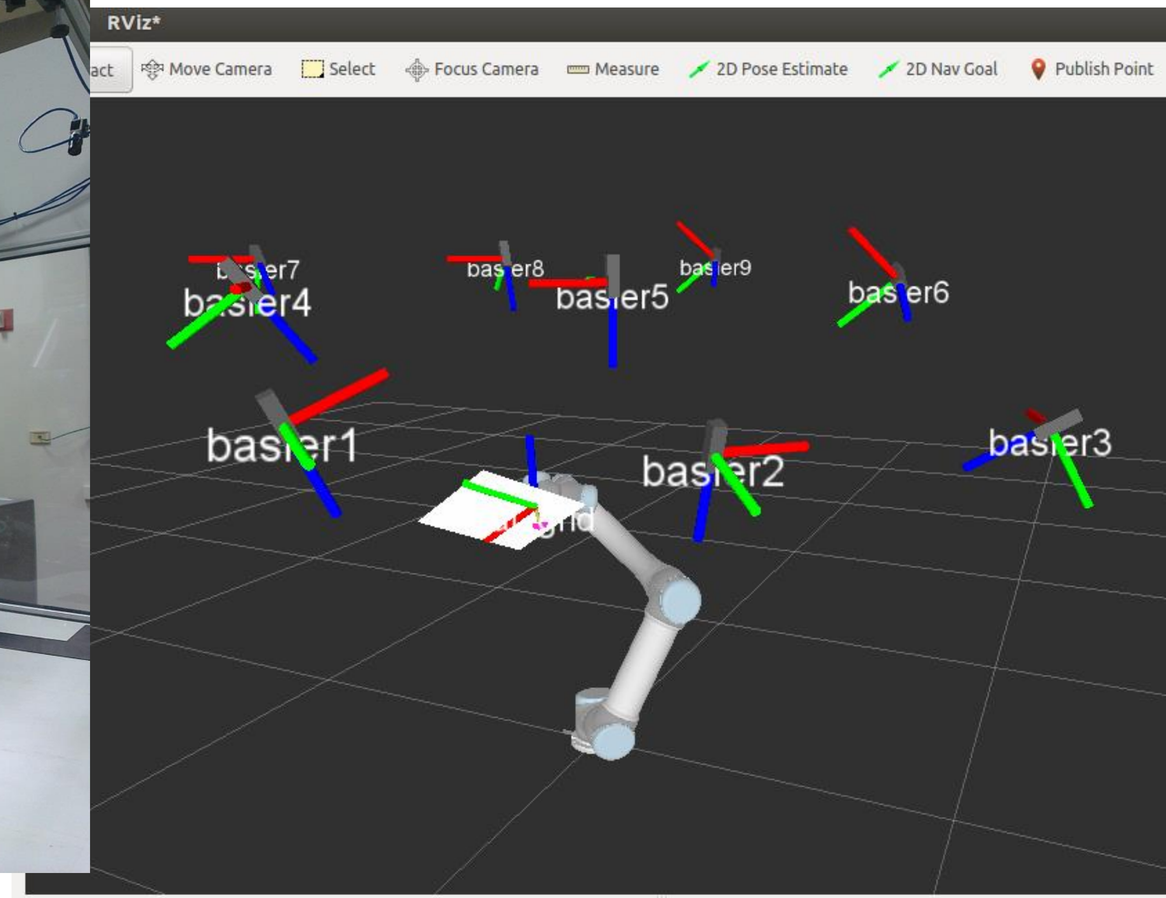
- Perform intrinsic and extrinsic calibration
- Continuously improving library
- Resources, library
  - Github link
  - Wiki link
- Resources, tutorials
  - Github industrial calibration tutorials link
  - Training Wiki link

Industrial (Intrinsic) Calibration



- The new 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



Industrial (Extrinsic) Calibration

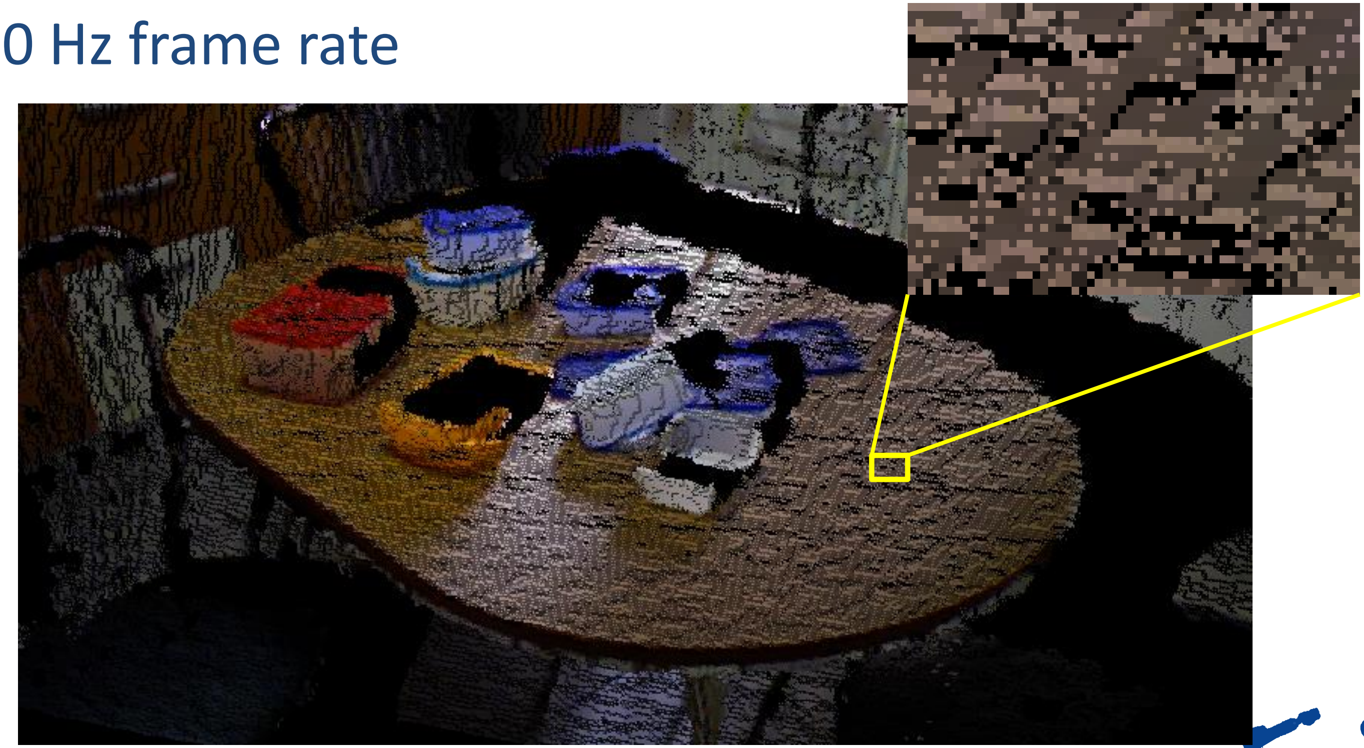
- https://www.youtube.com/watch?v=MJFtEr_Y4ak

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 openni_launch or openni2_launch
- Driver for Kinect 2.0 is in package iai_kinect2 ([github link]())
- http://rosindustrial.org/news/2016/1/13/3d-camera-survey

3D Cameras

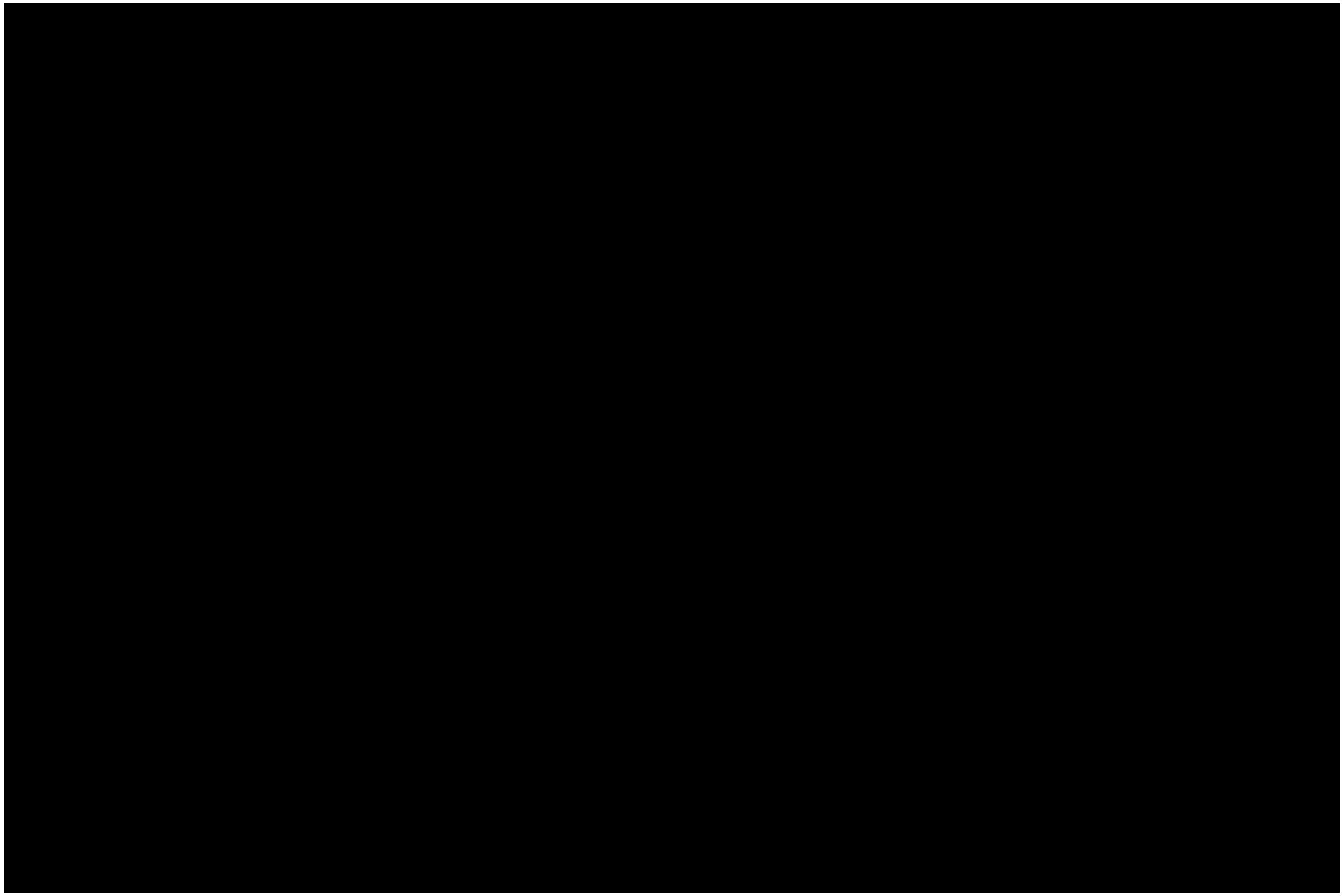
- Produce (colored) point cloud data
- Huge data volume
  - Over 300,000 points per cloud
  - 30 Hz frame rate



Exercise 4.2

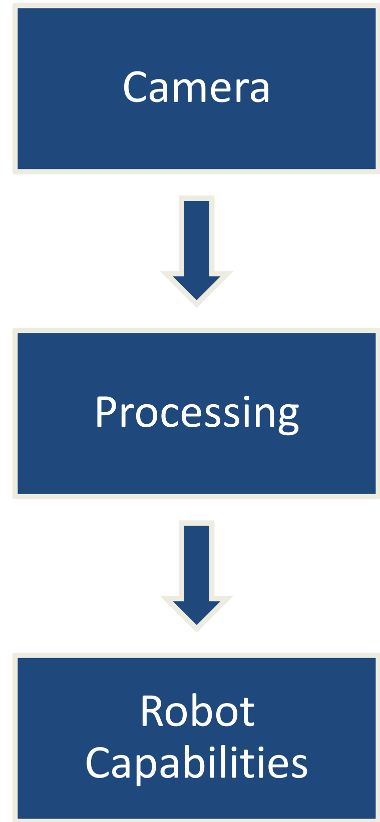
- Play with PointCloud data
  - bag file of pre-recorded Asus data
  - Matches scene for demo_manipulation
  - 3D Data in ROS
- https://github.com/ros-industrial/industrial_training/wiki/Introduction-to-Perception

Example: Pick & Place



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



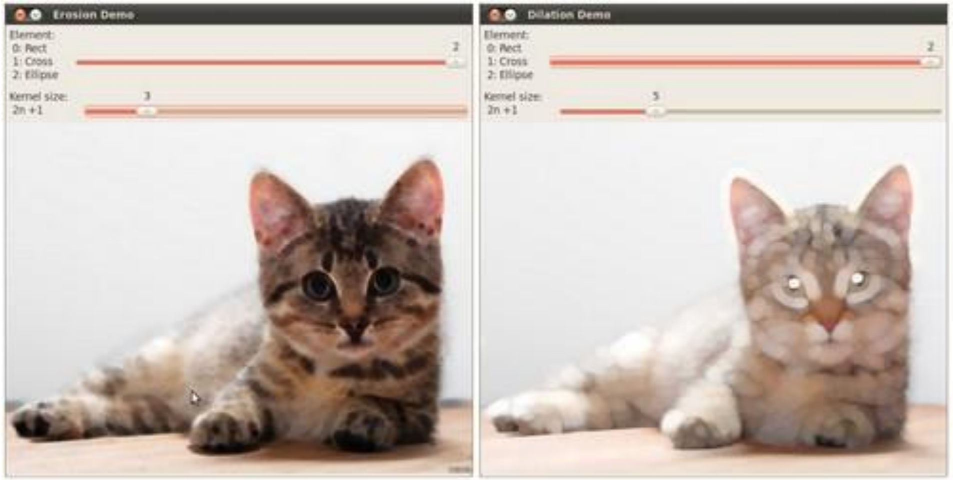
Perception Tools

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

- Focus on PCL tools for exercise

Perception Libraries (OpenCV)

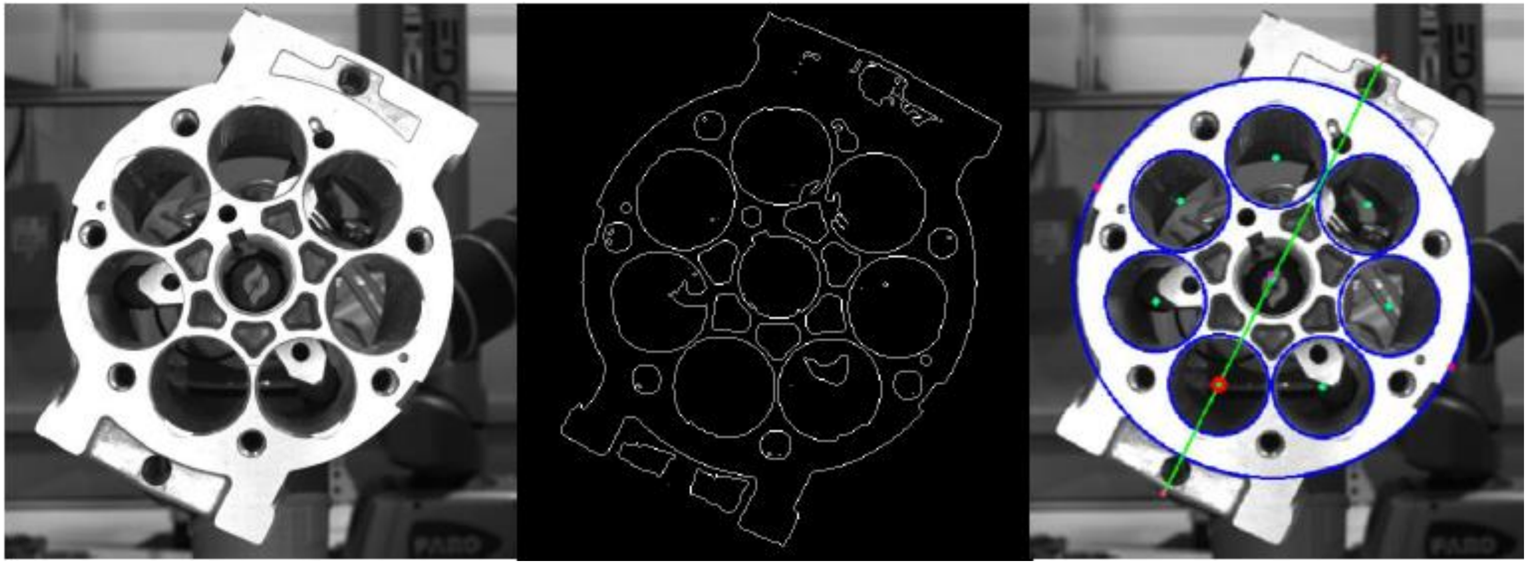
- Open Computer Vision Library (OpenCv) - http://opencv.org/
  - Focused on 2D images
  - 2D Image processing
  - Video
  - Sensor calibration
  - 2D features
  - GUI
  - GPU acceleration



http://opencv.org

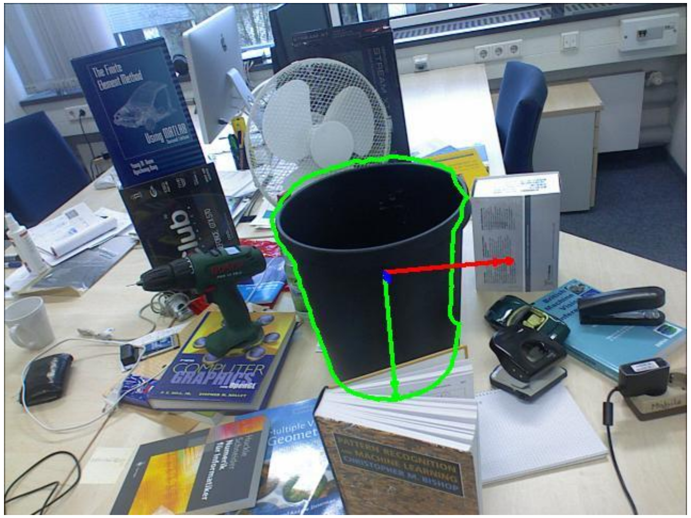
OpenCV tutorial

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



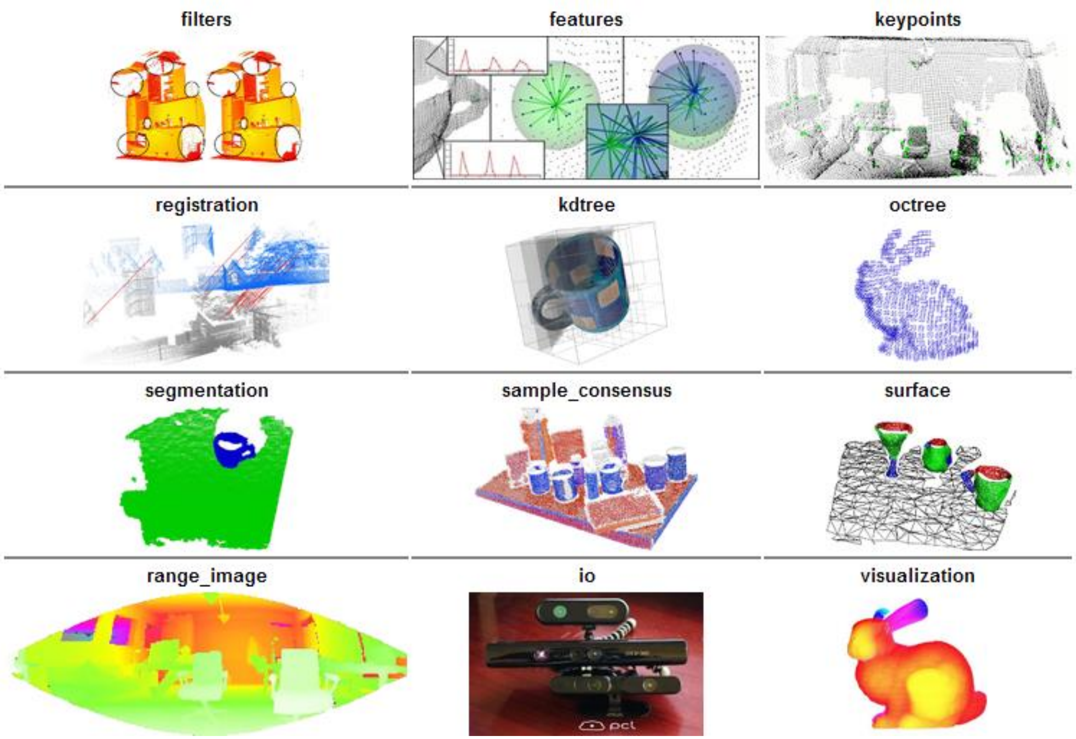
Perception Libraries (OpenCV)

- Open CV 3.2
  - Has more 3D tools
    - [LineMod](https://www.youtube.com/watch?v=vsThfxzIUjs)
      - 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

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
  - Object Recognition
    http://www.ros.org/wiki/object_recognition
  - Robot Self Filter
    http://www.ros.org/wiki/robot_self_filter

Perception Pipeline

Overall Process

- 3D Camera
- Processing
- Robot Capabilities

Perception Process

- Obtain PointCloud
- Convert PointCloud ROS->PCL
- Filter PointCloud
- Convert PointCloud PCL->ROS
- Publish PointCloud
- Broadcast Transform*

PCL Methods

- Voxel Grid
- PassThrough
- RANSAC Plane Segmentation
- Others...

Lab

- Exercise 4.3 - https://github.com/ros-industrial/industrial_training/wiki/Building-a-Perception-Pipeline

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

Descartes

- Path Planning
- Trajectory points
- Robot model representation

Perception

- Calibration
- PointCloud Bag File
- OpenCV
- PCL