Tool Path Planning Workshop

October 21, 2024

Agenda

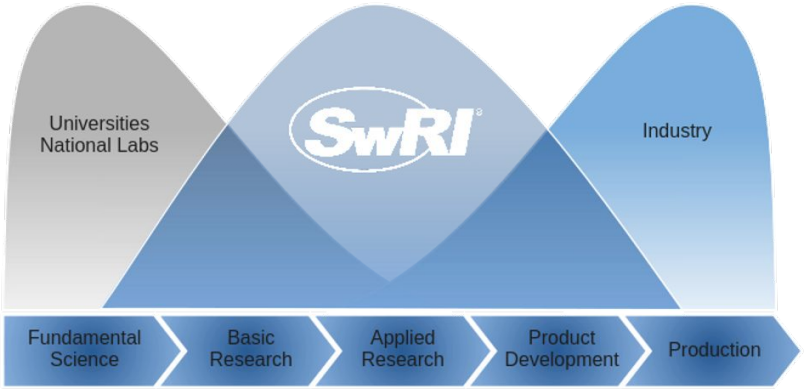
- 13:00 - Introductions/Logistics
- 13:10 - Tool path planning/Noether repository
- 13:45 - Exercise 0
- 14:00 - Exercise 1a
- 14:30 - Exercise 1b
- 15:00 - Break
- 15:15 - Exercise 2a
- 15:45 - Exercise 2b
- 16:15 - Exercise 2c
- 16:45 - Q&A

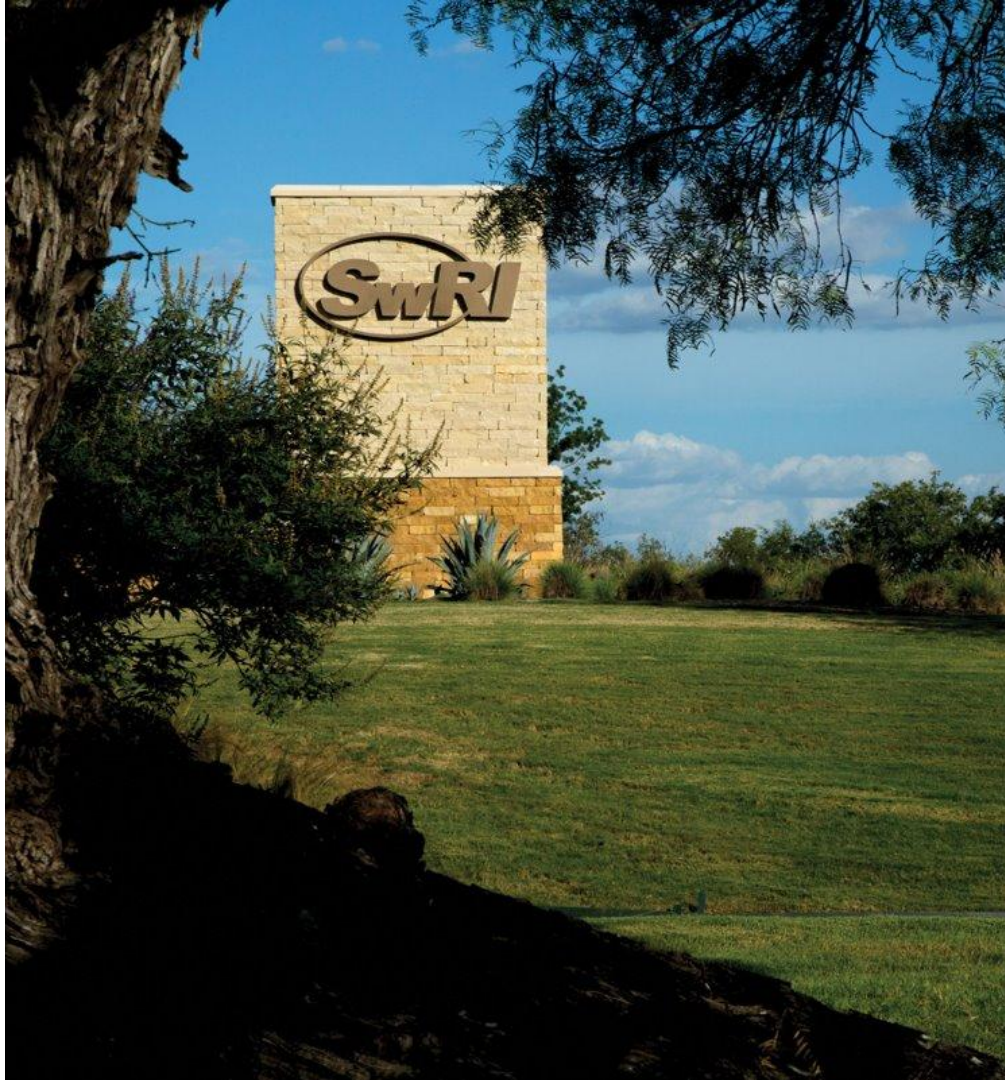
Logistics

- WiFi
  - Network: comwellhotels
  - Password: comwellhotels
- Restrooms
- Coffee/refreshments
- Repository
  - https://github.com/marip8/noether_roscon_2024

Introductions

- SwRI
  - Founded in 1947
  - San Antonio, TX
  - Independent, not-for-profit
  - Applied R&D in Natural Sciences and Engineering
- SwRI Robotics
  - Vehicle autonomy (off-road, on-road)
  - Perception driven manipulation
  - Custom robotics





Goals

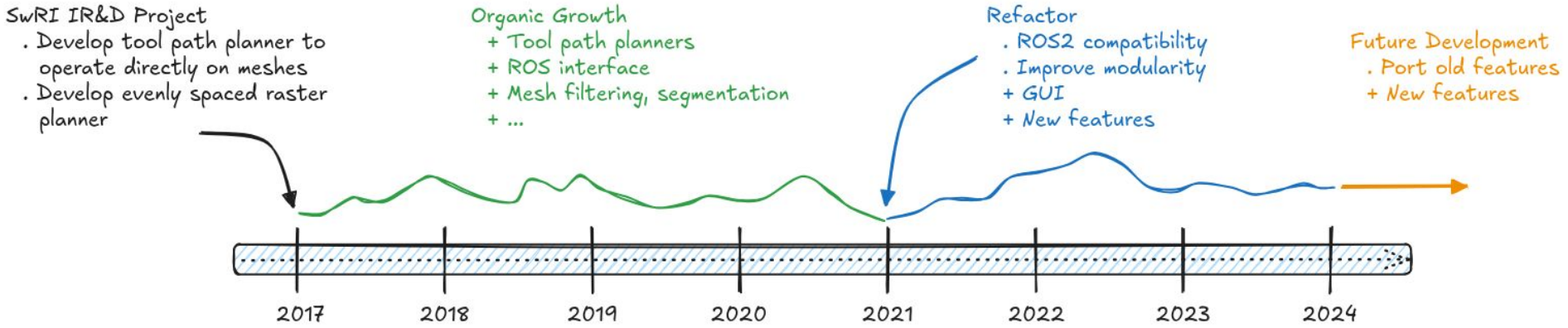
- Become familiar with tool path planning capabilities in Noether
- Understand how to create desired tool paths using the existing capability modules
- Provide reference material for future tool path planning applications
- Understand how to write custom Noether plugins

Tool Path Planning

- What is tool path planning?
- Difference from motion planning?
- Existing tools
  - CAD/CAM, OpenCascade, Offline Programming Tools
  - Limitation: CAD geometry (e.g., B-Rep surfaces) required
- What about scanned parts?
  - Mesh/point cloud to CAD?
  - CAD model alignment?

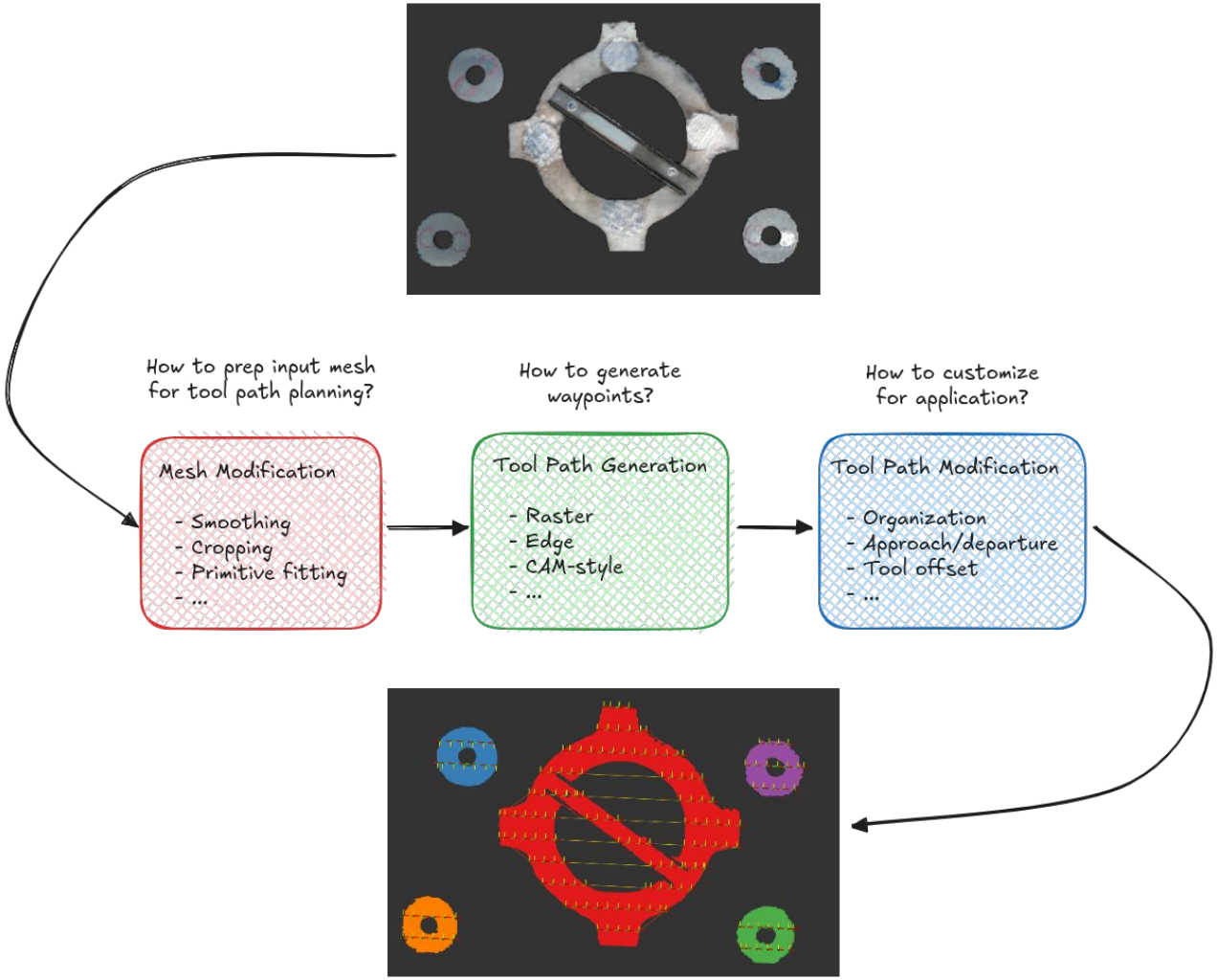
Noether

- Repository for tool path planning
- github.com/ros-industrial/noether
- History

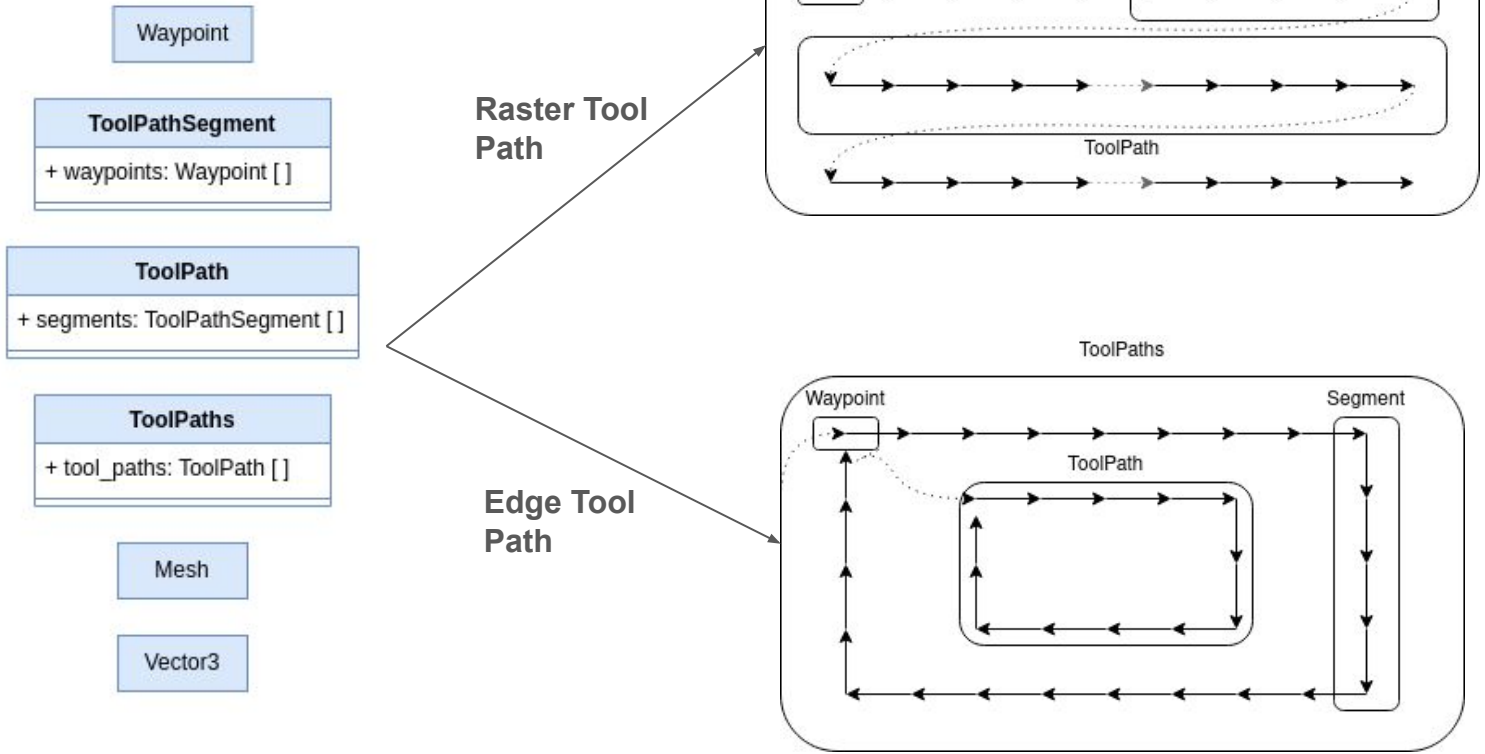


Noether

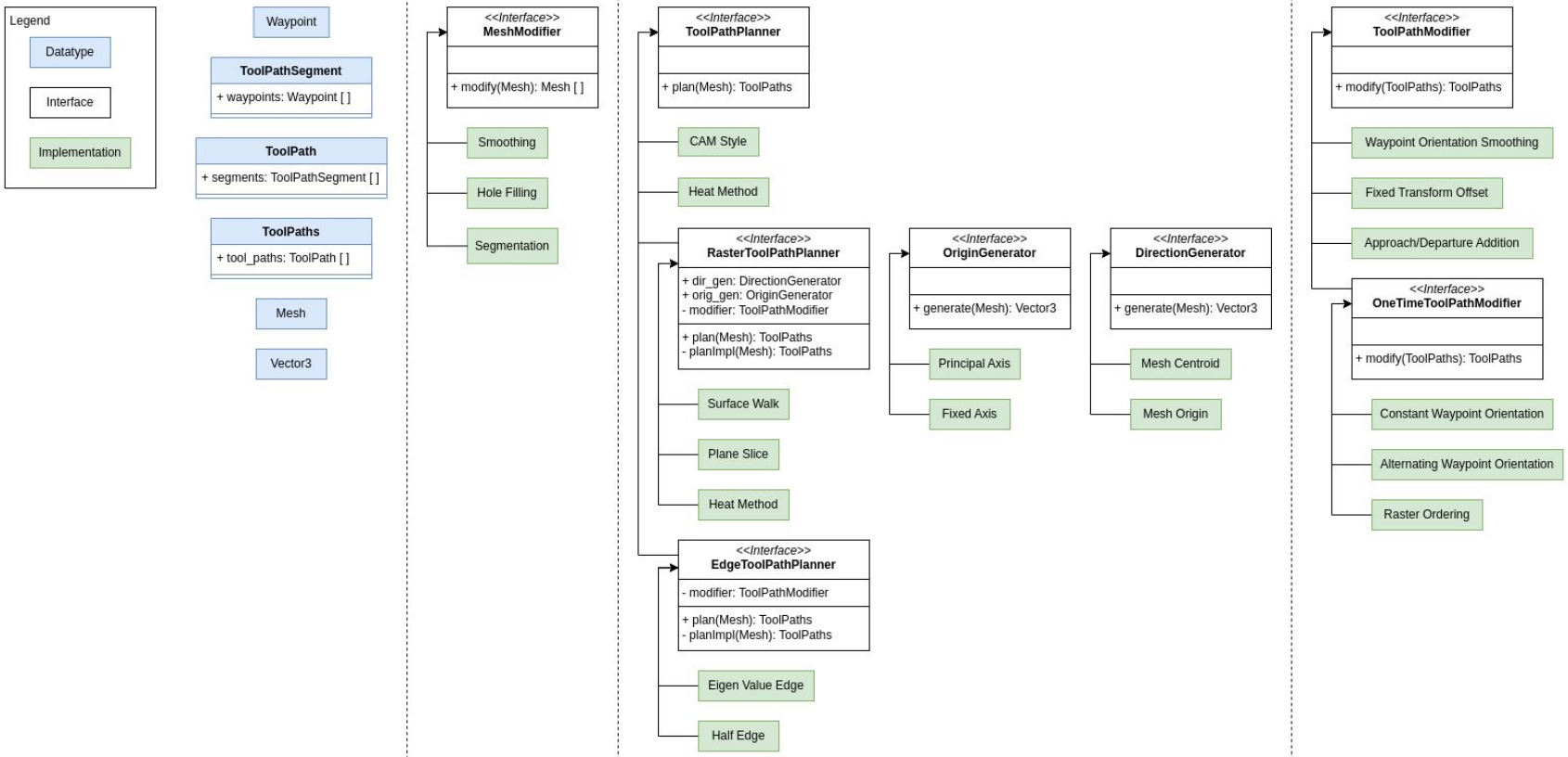
- Pipeline
- Concepts
  - 3 general TPP tasks
  - Base classes for each task



Noether

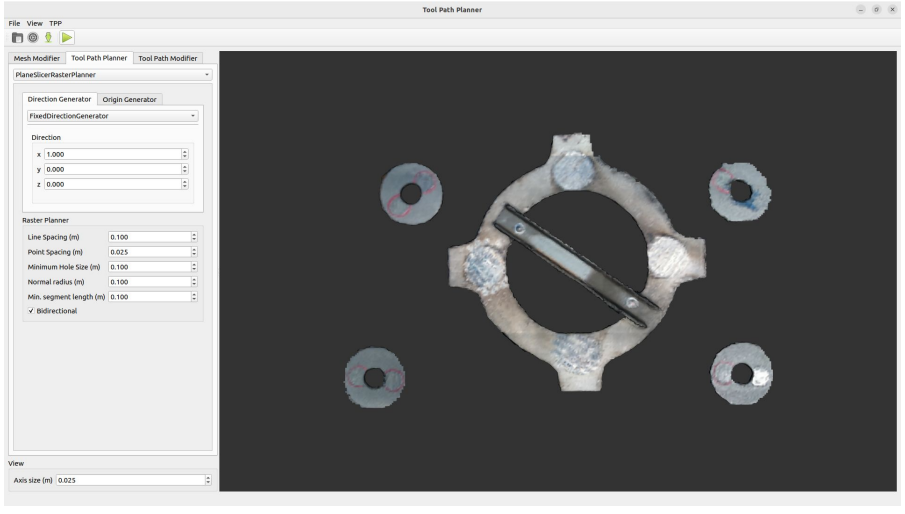

Waypoint
ToolPathSegment
+ waypoints: Waypoint [ ]
ToolPath
+ segments: ToolPathSegment [ ]
ToolPaths
+ tool_paths: ToolPath [ ]
Mesh
Vector3
Raster Tool Path
Edge Tool Path
ToolPaths
Waypoint
Segment
ToolPath
ToolPaths
Waypoint
Segment
ToolPath

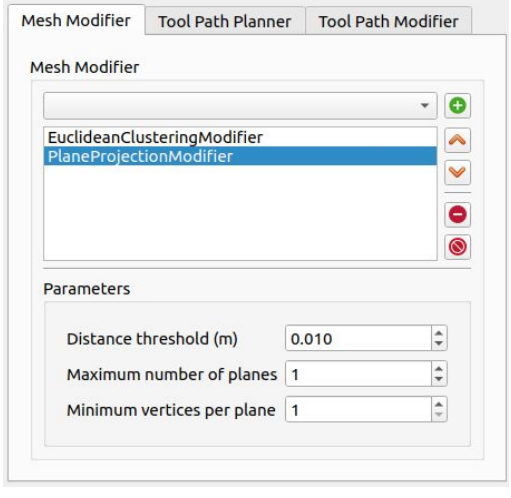
Noether

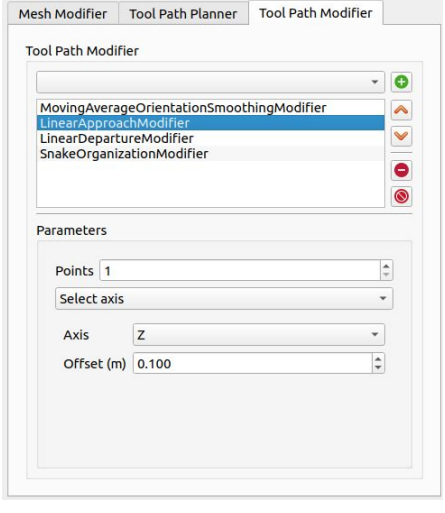

Legend
Datatype
Interface
Implementation
Waypoint
ToolPathSegment
+ waypoints: Waypoint [ ]
ToolPath
+ segments: ToolPathSegment [ ]
ToolPaths
+ tool_paths: ToolPath [ ]
Mesh
Vector3
<<Interface>>
MeshModifier
+ modify(Mesh): Mesh [ ]
Smoothing
Hole Filling
Segmentation
<<Interface>>
ToolPathPlanner
+ plan(Mesh): ToolPaths
CAM Style
Heat Method
<<Interface>>
RasterToolPathPlanner
+ dir_gen: DirectionGenerator
+ orig_gen: OriginGenerator
- modifier: ToolPathModifier
+ plan(Mesh): ToolPaths
- planImpl(Mesh): ToolPaths
Surface Walk
Plane Slice
Heat Method
<<Interface>>
EdgeToolPathPlanner
- modifier: ToolPathModifier
+ plan(Mesh): ToolPaths
- planImpl(Mesh): ToolPaths
Eigen Value Edge
Half Edge
<<Interface>>
OriginGenerator
+ generate(Mesh): Vector3
Principal Axis
Fixed Axis
<<Interface>>
DirectionGenerator
+ generate(Mesh): Vector3
Mesh Centroid
Mesh Origin
<<Interface>>
ToolPathModifier
+ modify(ToolPaths): ToolPaths
Waypoint Orientation Smoothing
Fixed Transform Offset
Approach/Departure Addition
<<Interface>>
OneTimeToolPathModifier
+ modify(ToolPaths): ToolPaths
Constant Waypoint Orientation
Alternating Waypoint Orientation
Raster Ordering

Noether GUI

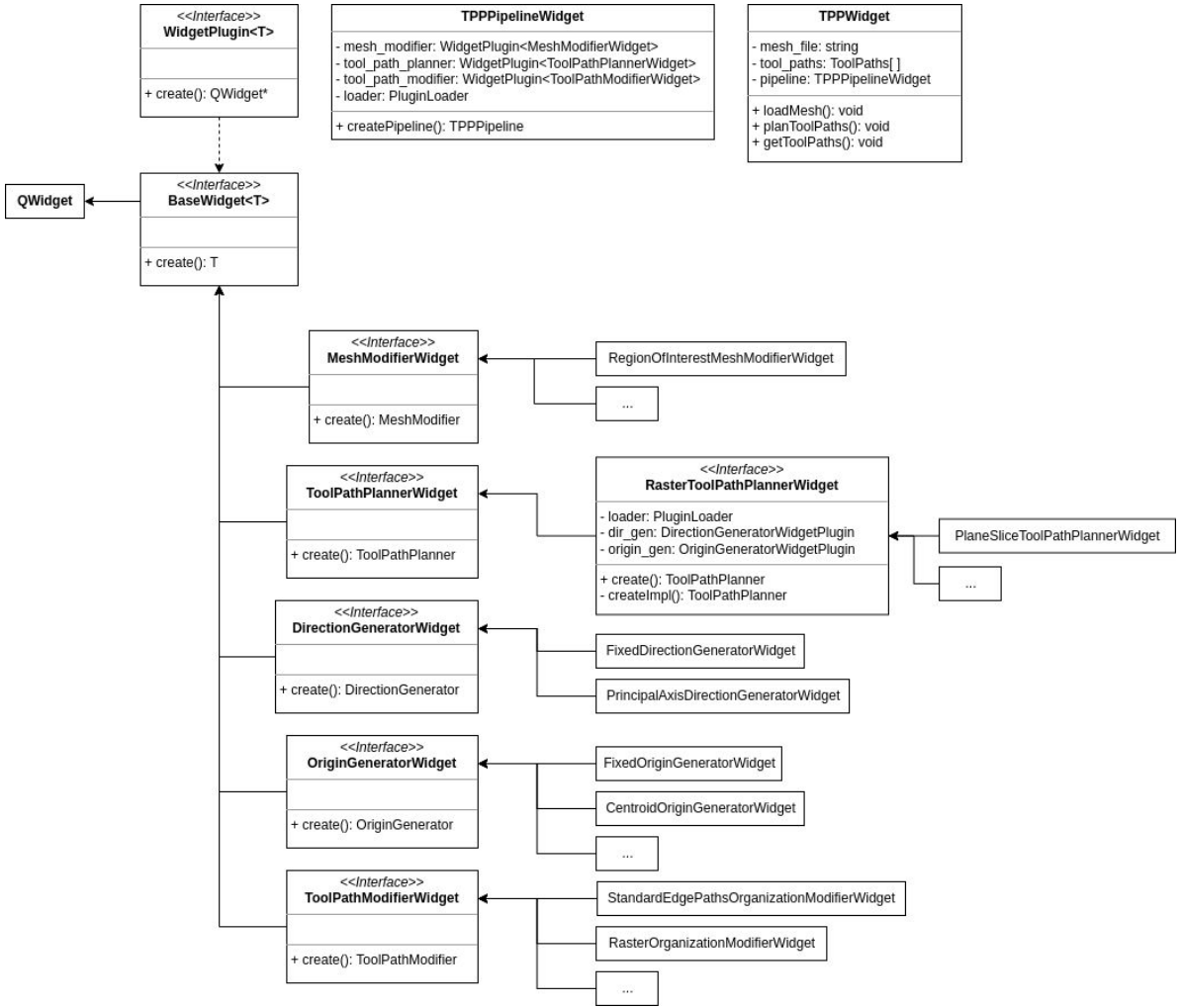
- GUI
  - Allow users to configure/test TPP pipeline
- TPP elements are plugins
  - Plugin defines widget UI for TPP element
  - Widget UI configures TPP element
  - GUI constructs TPP pipeline from activated plugins
- Why plugins?
  - Modularity
  - Compatibility with proprietary code







Noether GUI


<<Interface>>
WidgetPlugin<T>
+ create(): QWidget*
TPPPipelineWidget
- mesh_modifier: WidgetPlugin<MeshModifierWidget>
- tool_path_planner: WidgetPlugin<ToolPathPlannerWidget>
- tool_path_modifier: WidgetPlugin<ToolPathModifierWidget>
- loader: PluginLoader
+ createPipeline(): TPPPipeline
TPPWidget
- mesh_file: string
- tool_paths: ToolPaths[ ]
- pipeline: TPPPipelineWidget
+ loadMesh(): void
+ planToolPaths(): void
+ getToolPaths(): void
QWidget
<<Interface>>
BaseWidget<T>
+ create(): T
<<Interface>>
MeshModifierWidget
+ create(): MeshModifier
RegionOfInterestMeshModifierWidget
...
<<Interface>>
ToolPathPlannerWidget
+ create(): ToolPathPlanner
<<Interface>>
RasterToolPathPlannerWidget
- loader: PluginLoader
- dir_gen: DirectionGeneratorWidgetPlugin
- origin_gen: OriginGeneratorWidgetPlugin
+ create(): ToolPathPlanner
- createImpl(): ToolPathPlanner
PlaneSliceToolPathPlannerWidget
...
<<Interface>>
DirectionGeneratorWidget
+ create(): DirectionGenerator
FixedDirectionGeneratorWidget
PrincipalAxisDirectionGeneratorWidget
<<Interface>>
OriginGeneratorWidget
+ create(): OriginGenerator
FixedOriginGeneratorWidget
CentroidOriginGeneratorWidget
...
<<Interface>>
ToolPathModifierWidget
+ create(): ToolPathModifier
StandardEdgePathsOrganizationModifierWidget
RasterOrganizationModifierWidget
...

Exercises Overview

- Build the repository
- Use existing modules to create specific tool paths
- Customize the UI for an existing capability module
- Create custom capability module with UI
- Create specific tool path using custom components

Exercise 0

- Build the workspace
- https://github.com/marip8/noether_roscon_2024
- Docker Option
  - Build from `ros:humble` Docker image, or:
  - Pull and run pre-built Docker image at end of exercise 0
    - `docker login ghcr.io`
    - `cd <repository>/docker`
    - `docker compose up`

Exercise 1

Experiment with the Noether GUI!

- Tasks
  1. Load a mesh
  2. Create a raster tool path over the whole mesh
  3. Create a raster tool path over each component of the mesh
  4. Create a raster tool path over each component of the mesh, where the tool paths are generated on a plane rather than the mesh surface
  5. Modify the output tool path such that the x-axis of the waypoints flips from one stroke of the raster to the next
  6. Add an approach and departure point above the start/end of each raster
  7. Visualize the tool path lines and modified mesh

Exercise 1b

- Create a tool path for edge deburring on the puzzle piece
  - puzzle_piece.ply
  - puzzle_piece_refined.ply
- Constraints
  - Z-axis facing “up”
  - Even point spacing
- How might you handle the 2.5D nature of the part?
  - Hint: it’s hacky for this exercise
- How could we formalize the approach to be more exactly what we want?

Break

Problem Statement

- Perform camera-based inspection of cylindrical components
- Tasks
  a. Create a mesh modifier to extract objects that look like cylinders
  b. Generate raster tool paths
  c. Offset the tool paths to a valid camera position

Exercise 2a

- Create a new UI to configure an existing module that can transform waypoints on the mesh surface into camera viewpoints
  - Rotate such that z-axis points at the surface
  - Apply standoff
- Which module to customize?
- Tasks
  - Create widget that can produce and configure a `OffsetModifier`
  - Add UI elements to the widget for the user to input configuration information
  - Create a plugin from this widget
  - Test

Exercise 2b

- Create a mesh modifier to extract objects that look like cylinders
  - RANSAC primitive fit using PCL
  - Project inliers onto fitted model
- Tasks
  - Create a mesh modifier that extracts cylindrical sub-meshes
  - Create a widget to create and configure this mesh modifier
    - What information do you want to share close to the user?
  - Create a plugin for this widget
  - Test

Exercise 2c

- How to test modifier components?
- GUI requires a tool path planner??
- Easy hack: make a planner that doesn’t do anything!
- Tasks:
  - Create a tool path planner object that does nothing
  - Create a widget to create and configure this tool path planner
  - Create a plugin for the widget
  - Test

Putting it all together

- Review
  - Noether repository for tool path planning
  - Pipeline
  - Can customize behavior **and/or** UI
  - Think modularly; keep components simple
- Looking forward
  - Where is the ROS/ROS2 interface?
  - Docker image?
  - Tagged releases?
- Engagement welcome
  - Contributions
  - Issues
  - Feature requests

Q&A

Thanks for attending!

Michael Ripperger
Southwest Research Institute
michael.ripperger@swri.org
@marip8