Overview

This project is designed to eliminate the need for a human to manually clean parts stripped from military vehicles. Instead, a sprayer will be attached to an articulated robotic arm. This arm will receive a path based on a 3D model of the part built by a scanner each time a new part is brought in. The path will then be subject to review by a technician using a graphical interface. After the cleaning is performed, the technician will then be able to check that the part is sufficiently clean and indicate the tool to perform more cleaning as necessary.

Parts of the Project

The project is broken down into a few different components: the path planner, the impingement modeler, the simulation, and the user interface. These components are hooked together using ROS (Robot Operating System).

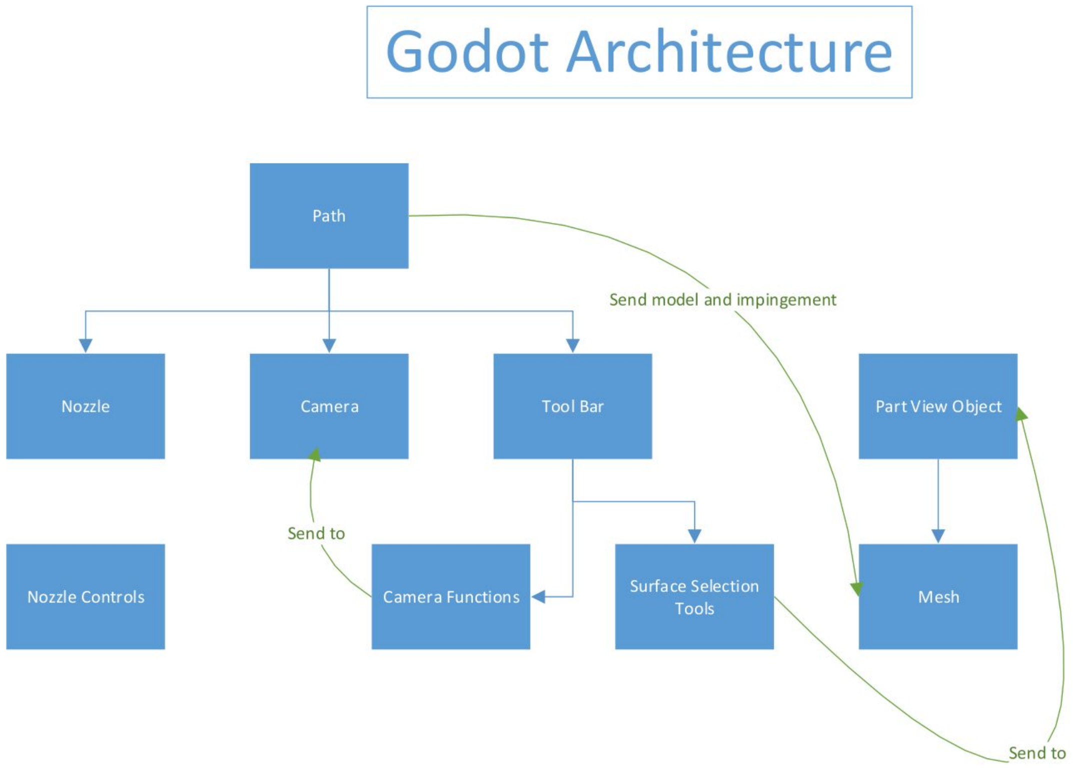
The path planner inputs the part model and creates a set of points along which the robot will follow to apply spray to the part. This process is kicked off by the user interface, and is the first task to be accomplished once a part model has been selected because the generated path is used by each of the other components.

The impingement modeler uses the part model and the generated path to simulate the intensity of spray coverage on each facet of the part model. This data is used to give the user an idea of how much attention each section of the part has gotten after the tool has completed the task. This could also be used later as a component of path generation to automatically give the path planner feedback without human intervention.

The simulation uses the generated path and the constraints of the robot itself to perform the inverse kinematics necessary for the robot to place the sprayer nozzle along the path at and in between each path position. This should also display to the user a model of the robot following the path.

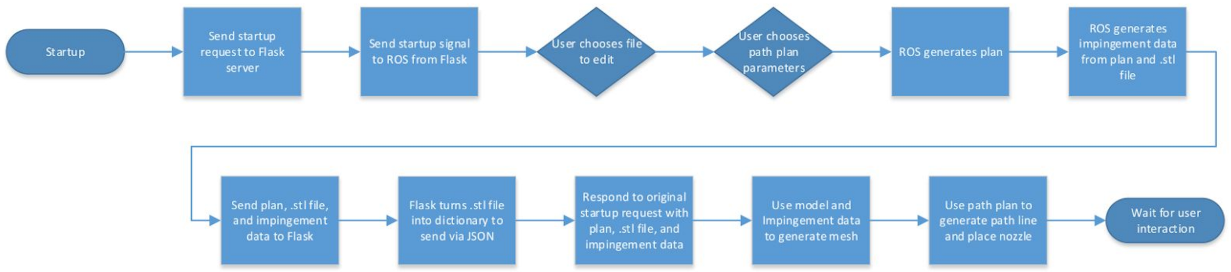
The user interface displays to the user the part model, the impingement model, and the generated path. This allows the user to get an idea of where the sprayer will move and the coverage of the part in 3D space. The user will then be able to select facets of the part model to receive more attention and send that back to the path planner to generate a new iteration.

Design

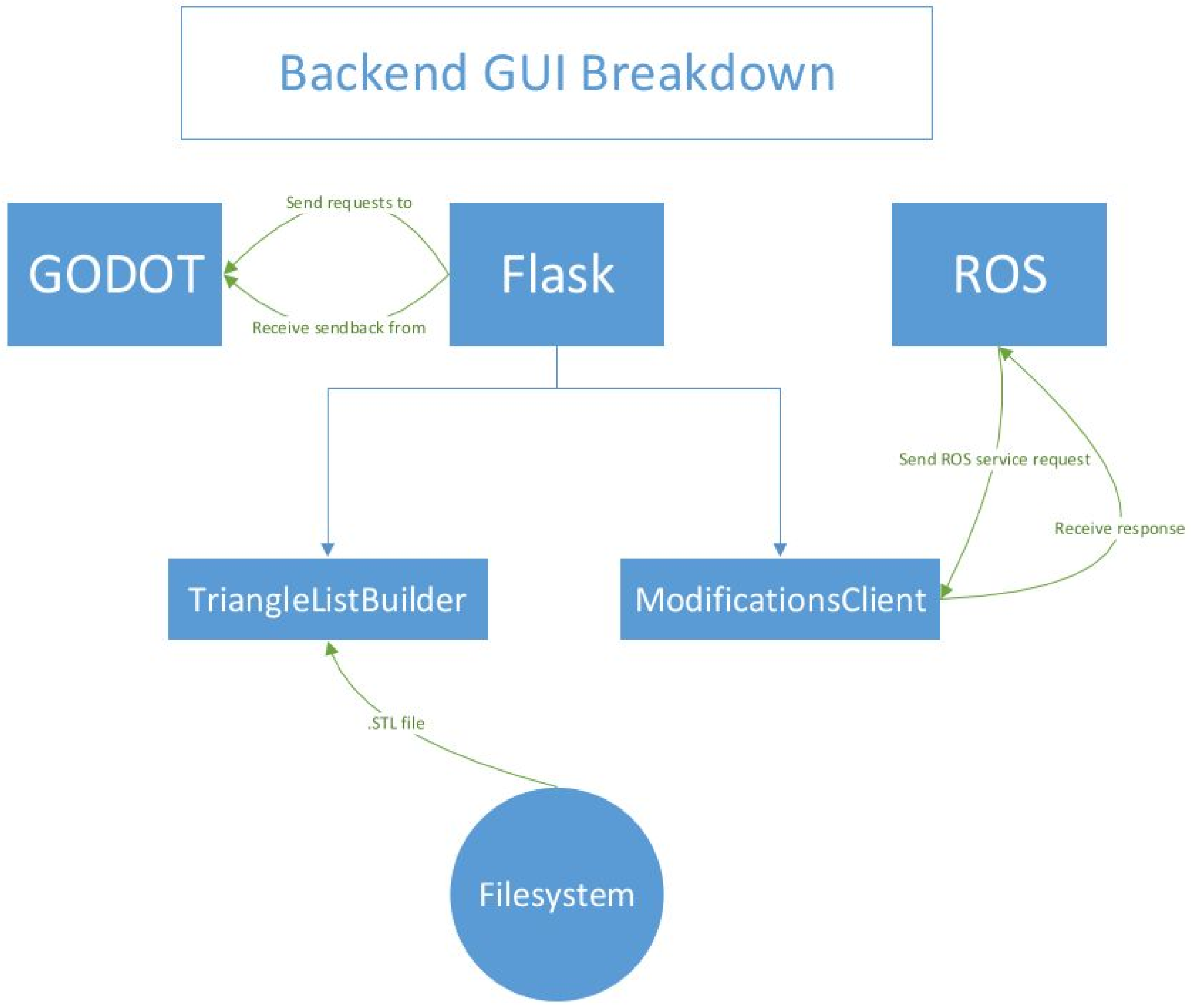


This is a broad view of the architecture in just the GODOT program. The blue lines indicate class hierarchy, while the green lines indicate communication channels. The communication to the back end is not shown here, but it comes from the Path object.

Startup Flow


Startup
Send startup request to Flask server
Send startup signal to ROS from Flask
User chooses file to edit
User chooses path plan parameters
ROS generates plan
ROS generates impingement data from plan and .stl file
Send plan, .stl file, and impingement data to Flask
Flask turns .stl file into dictionary to send via JSON
Respond to original startup request with plan, .stl file, and impingement data
Use model and Impingement data to generate mesh
Use path plan to generate path line and place nozzle
Wait for user interaction

This shows the startup sequence from when the user interface starts up until when the user can interact with the screen.



This is the breakdown of the Flask server that serves as an intermediary between the GODOT front end and ROS. The blue lines indicate class hierarchy, while the green lines indicate communication channels.

Chronology of Changes

This project has been subject to a few updates to our design goals. Here I will attempt to cover the main three shifts from my perspective as the GUI designer.

The first iteration of the project involved trying to figure out how we would present the data to the user in such a way as to allow them to view the part model, the path, and an interface through which to modify the path. Since the team was currently using RViz to display the simulation, I thought that it would be best to attempt to put as much of the user interface into RViz as possible.

To accomplish this, I made a Python QT UI with buttons that interfaced with RViz to manipulate nodes in the visualizer to show a nozzle moving along the path. While this worked in a limited way, it became clear that our design goals were limited by RViz. In order to allow the user to manipulate the path in a more intuitive way, we thought that the user would have to be able to click on the path itself, which was not an option within RViz.

This kicks off the stage when we attempted to give the user the ability to select the path and manipulate it manually. To give us more freedom, I started looking for an OpenGL based framework that would meet our needs without increasing development time to an unmanageable degree.
After considering many frameworks, I landed on the Godot Engine. This provides a simple scripting language similar to Python and a graphical interface to build interface buttons and place static meshes. After getting the part model and path into the display window, the first idea I had for making the path selectable was to generate objects at each of the points along the path where the nozzle changed direction, and allow the user to select these points to allow the user to operate on the line segment in between the points. This proved to be a little clunky, so we decided to create line segment object to select instead of points.
Once the user had a segment selected, they were presented with a nice, complicated set of instructions to modify the rotation of the nozzle, number of passes, speed, and more for that specific section of the path. After review, it was determined that the technician should not have this level of control over the path, so we pivoted once more.

The new plan was to allow the user to select an area on the part that they see needs more attention. This eliminates a huge amount of human guess work, and ends up with a much cleaner looking interface.