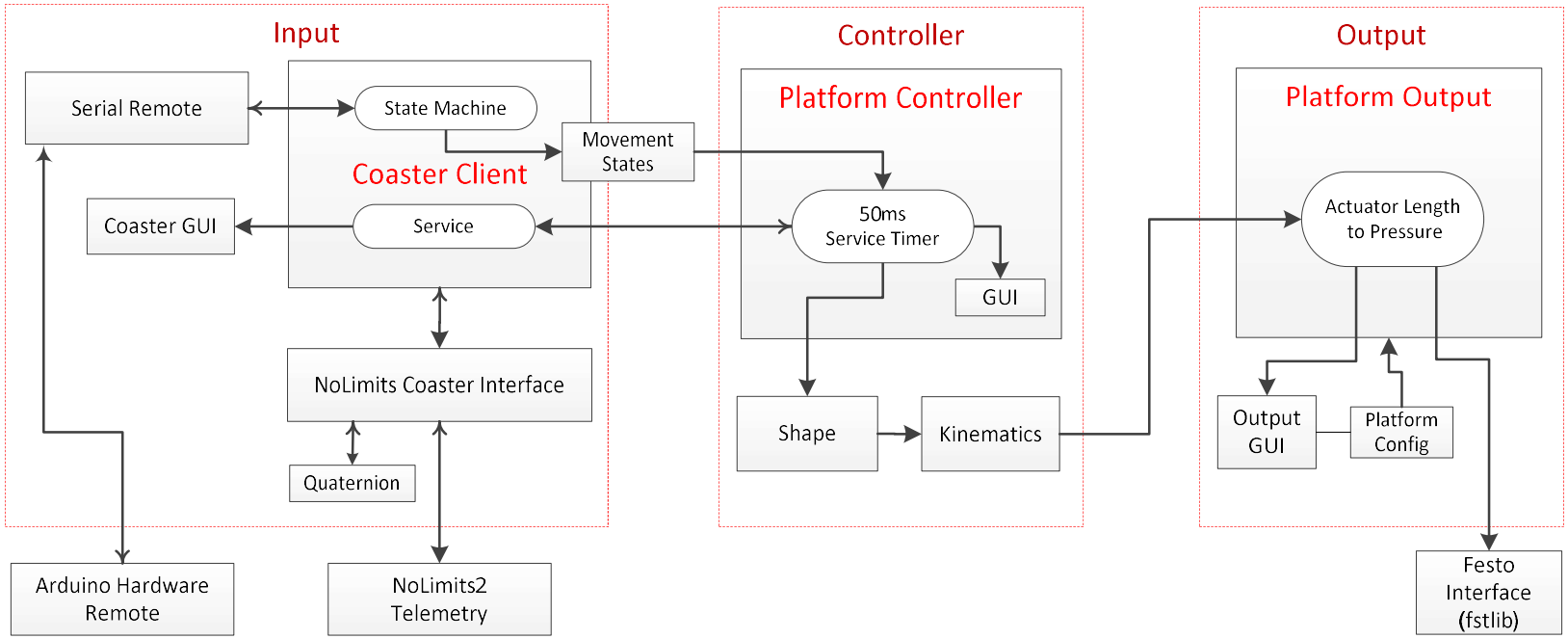
Software Overview

- Input from any 6DOF simulator
  (currently running with: NoLimits2, FlyInside, Prepar3D, X-Plane)
- Adjustable gain, washout and smoothing
- GUI and physical console ride control
- UDP or Serial output of actuator lengths
- Real time input and output status display
- All software open source, written in python

Software Technical Overview



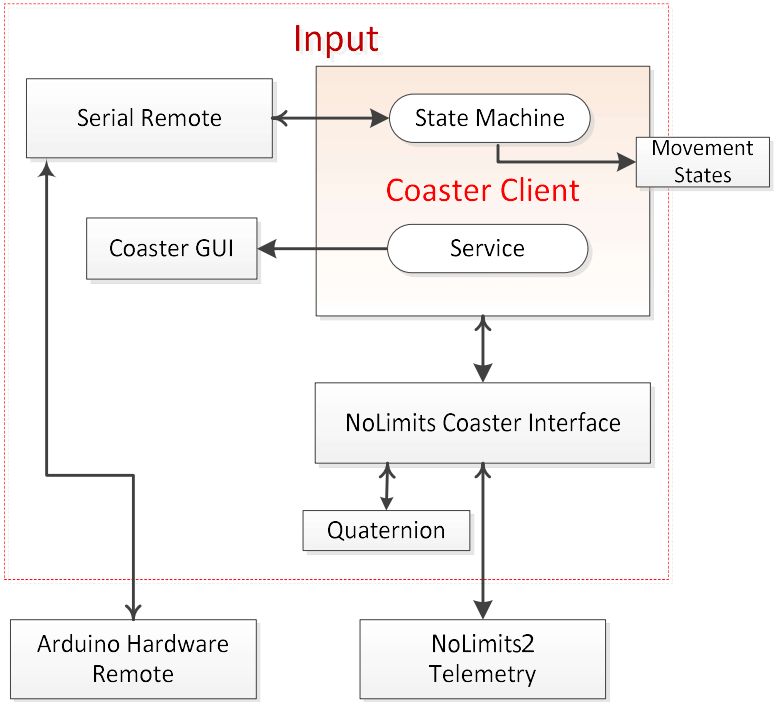
The platform controller pulls orientation requests from the coaster client 20 times per second.

Client responds with current orientation (optionally with commands to activate, deactivate or position the platform for access).

Orientation requests ‘shaped’ for gain & washout, converted to physical lengths using inverse kinematics

Actuator lengths sent to platform output module for conversion to pressure and sent to Festo controller

Coaster Client



Coaster_client.py

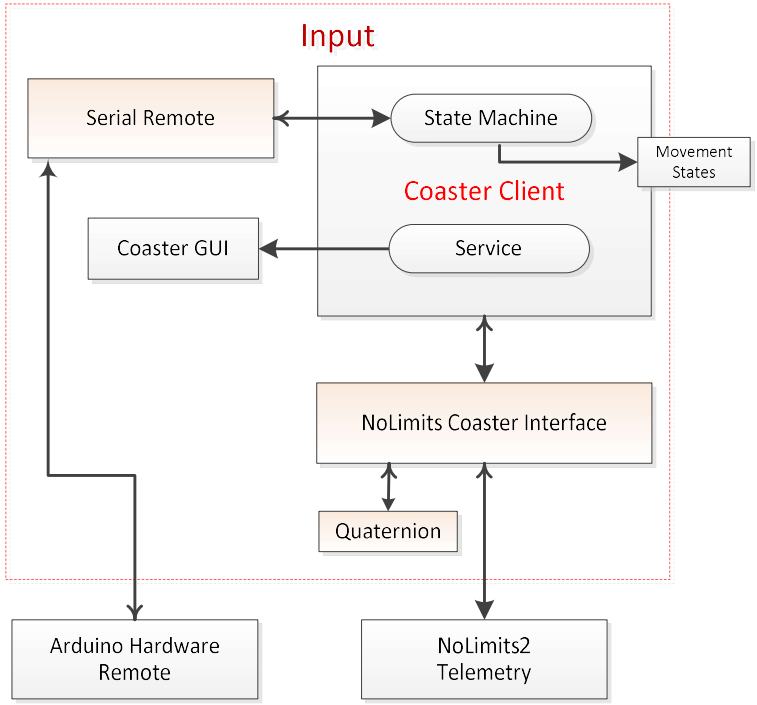
Service method is polled 20 times per second:
Serial Remote is polled for one of the following commands:
Activate – enable the platform to move
Deactivate – disable platform movement (Emergency stop if running)
Dispatch – start the ride
Pause – pause the ride (or un-pause if already paused)
Reset – return coaster to station if emergency stopped

Telemetry data requested from coaster interface, this data includes:
coaster speed, pause mode, surge, sway, heave, roll, pitch , yaw

Coaster state machine is updated based on following events:
Activated – Activation switch is on
Disabled – Activation switch is off
Stopped (coaster not paused with speed zero)
Dispatched (coaster has left the station)
Paused – pause command received while running
Un-paused - pause command received while paused
Emergency stopped – Activation switched off while running
Resetting – coaster is returning to station after estop

Telemetry orientation data sent to Controller module `move_function`

Coaster Interface



Coaster_interface.py

`get_telemetry` method called by client:
telemetry request message sent to NoLimits:
roll, pitch and yaw calculated using Quaternion method in quaternion.py
yaw rate is calculated using yaw angle change from previous message
all values are limited to range of -1 to +1
returns: flag indicating pause state, speed, list of: surge, sway, heave, r, p, y

Methods to control coaster using telemetry:
set manual mode – stops auto run of coaster (used at startup only)
dispatch
Methods to control coaster using Windows keystrokes:
toggle pause
open harness, close harness, disengage floor
set speed (used to run coaster back to station after estop)

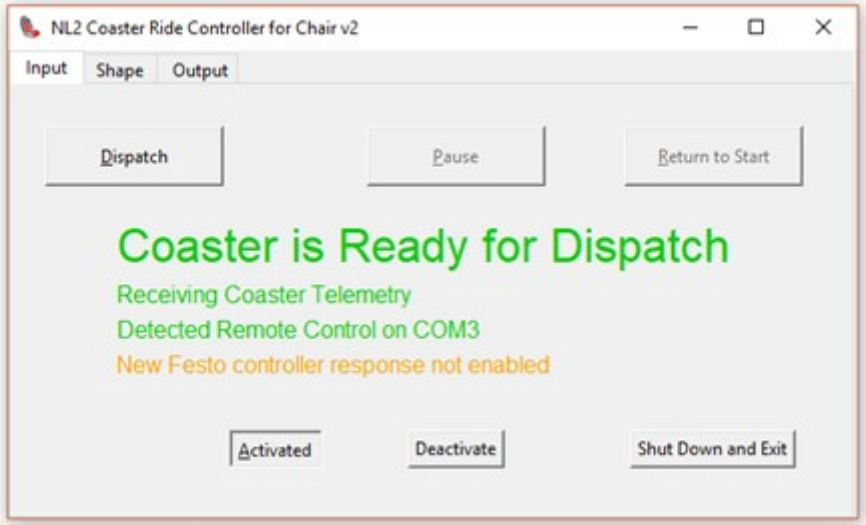
Quaternion.py
returns roll, pitch, and yaw from quaternion

Serial_remote.py
Auto detects remote control (looks for keyword response to probe string)

Sends one of following commands to client:
activate, deactivate, dispatch, pause, reset

Enum of coaster state is sent to remote when state changes

Coaster GUI



If NoLimits Window not found at start, displays messagebox:
Coaster Software Not Found -
(start NL2 or maximize window if already started)

coaster_gui.py

Displays coaster status derived from state machine in coaster client:
- Coaster at Station but deactivated
- Coaster is Ready for Dispatch
- Coaster is Running
- Coaster is Paused
- Emergency Stop
- Coaster is returning to station (after emergency stop)

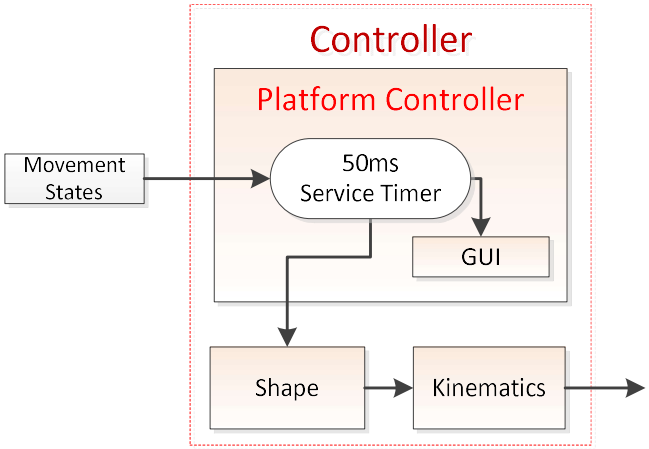
Displays NoLimits telemetry status:
- Receiving Coaster Telemetry
- Telemetry error + (message err string from NoLimits)
- Coaster Software Not Found

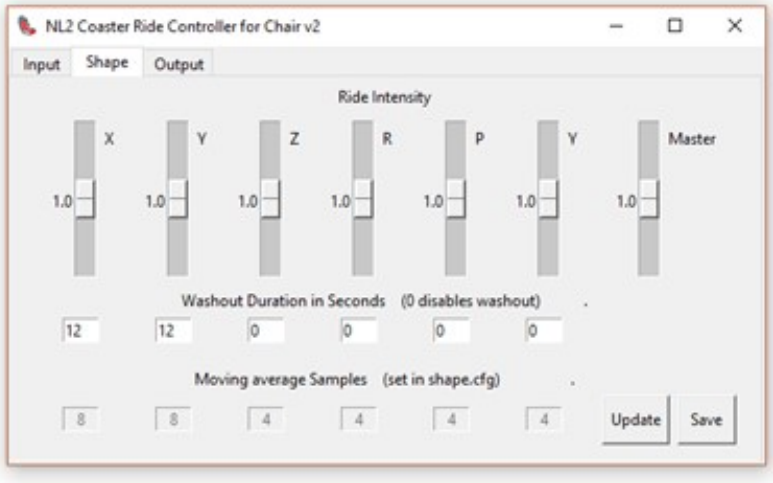
Displays remote controller status:
- Detected Remote Control on COMn

Or
- Looking for remote on COMn

Displays Festo communication status

Platform Controller





platform_controller.py

All system activity is derived from 50ms service frames controlled from this module

Each frame polls the client for platform position requests or commands to control the coaster.

Movement requests are sent to the shaper for control of gain, washout and smoothing.

After shaping, requests sent to kinematics.py for calculation of actuator lengths

This module contains the tk root and provides the UI for the Shape module

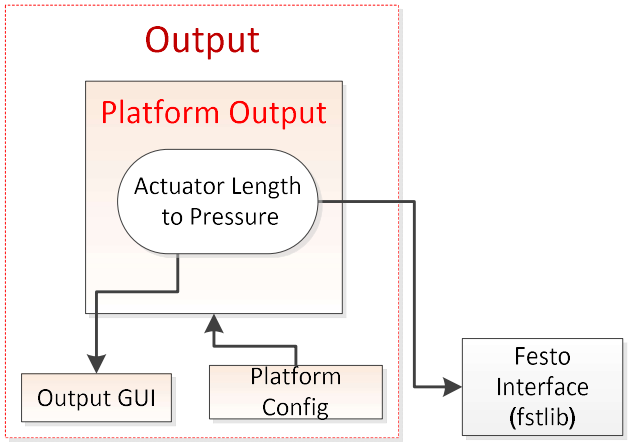
Shape.py

This module controls the gain for displacement and rotation values
Washout can be set to control the rate of decay for each value
Moving average allows smoothing of the values
This module also converts normalized input to real values in mm or radians

Kinematics.py

This module calculates the six actuator lengths for a Stewart platform that conforms to the input displacement and rotation values

Platform Output



platform_output.py

At startup, module calculates the actuator attachment points in 3D space using data from the platform config file. This information is accessed via get methods (get_geometry, get_actuator_lengths, get_limits)

At runtime, the move_platform method is called from the controller with required lengths of the six actuators.

If the platform is enabled, the pressure calculated that will contract each actuator to the desired length.

Pressures are sent to the Festo controller using UDP messages formed in the `_send_packet` method.

Output_gui.py

This module provides graphical and text information on platform orientation.

Six vertical bars indicate the length of each actuator.
A text box shows the following for each actuator:
- length in millimeters
- percent of movement is show in round brackets
- percent of error between desired pressure and actual pressure

The text goes red if requested movement exceeds possible movement

Top, front and side views of the chair move and rotate corresponding to the orientation request from the controller. The input request is displayed as text at the window bottom

Client Input Overview

At startup, clients are passed a callback method for movement requests and one for system commands

A service method is polled at regular intervals (20 times per second) for movement requests or system commands.

Movement requests have six parameters:

x,y,z,r,p,y

x is forward/back movement in millimeters (+ is forward)
y is side to side movement in mm (+ is left)
z is up/down movement in mm (+ is up)
r is roll in degrees (+ is left up)
p is pitch in degrees (+ is nose down)
y is yaw in degrees (+ is CCW)

Movement requests can be either actual values (mm and angles) or normalized (ranging from -1 to +1)

System commands can be one of the following:

Activate – enable the platform to move
Deactivate – disable platform movement (Emergency stop if running)
Dispatch – start the ride
Pause – pause the ride (or un-pause if already paused)
Reset – return coaster to station if emergency stopped

Keyboard, GUI and UDP Example Clients

All Clients provide position input using the following six parameters:
- x is forward/back movement in millimeters (+ is forward)
- y is side to side movement in mm (+ is left)
- z is up/down movement in mm (+ is up)
- r is roll in degrees (+ is left up)
- p is pitch in degrees (+ is nose down)
- y is yaw in degrees (+ is CCW)

All Clients can specify values as normalized(range from -1 to +1) or real values (mm and radians)
- constant `is_normalized` is set `True`, for narmalized, `false` for real values

All Clients can send the following command strings:
- "enable" - enables the platform to move
- "disable" – disables platform movement
- "exit" - terminates the software

Platform_input.py - keyboard input version that accepts comma separated values as: x,y,z,r,p,y

PlatformInputTk.py – GUI input using Tkinter
- x,y,z,r,p,y values are entered using sliders

PlatformInputUDP.py - receives UDP messages on port 10009
- Move messages are: "xyzrpy,x,y,z,r,p,y,\n "
- "xyzrpy " - header string preceding the request
- x,y,z (surge, sway, heave) are the three translation values in mm
- r,p,y (roll, pitch, yaw) are three rotation values in radians