Closeedloop_rt.py - Troubleshoot Summary & Next Steps

LAST UPDATED May 11, 2025

Jump to Section:

Issues with the original closedloop_rt.py

1. Missing Theta Normalization:

The original controller did not normalize theta values to the required [-1.7, 1.7] range. pFaces server rejects theta values outside this range, causing controller failures that were not identifiable due to lack of error handling.

2. Incomplete Hyperrectangle Formatting:

The get_hyper_rec_str function in LocalizationServerInterface only creates the first part of the required format and misses the fixed reference frame that pFaces uses as standardized computational domain.

The format returned:
{x_min,x_max},{y_min,y_max},{-3.2,3.2},{0.0,0.8}*/{-2.1,2.1}*
(object x and y coordinate bounds, calculated from OptiTrack server data, theta and velocity ranges (for targets/obstacles)

Required format:
{x_min,x_max},{y_min,y_max},{-3.2,3.2},{0.0,0.8}**|{2.0,3.0},{2.0,3.0},{-3.2, 3.2},{0.0,0.8}**
(fixed x, y coordinate bounds, same theta and velocity ranges as in first part)

3. Poor Error Handling:

Bare except blocks without specific error information, diagnostic messages, or fallback mechanisms when server connections fail. Resulted in empty responses.

4. Connection Management Issues:

No timeout handling for server connections or caching of data to reduce server requests. Connection reuse causing state management problems.

Improved Controllers

1. synth_and_test.py

The first test that successfully retrieved action commands from pFaces using hardcoded inputs. This verified basic communication between client and server.

2. final_controller.py

The first working live controller, executed from the Media Server PC.

RemoteSymbolicController.py file was also updated to improve reliability and integration.

Theta Handling Issue

After deployment, it was observed that theta values from the DeepRacer changed randomly, and the original closedloop_rt.py failed on some inputs. This led to the development of more robust controller versions.

3. robust_controller.py

- Adds theta normalization by clamping theta to [-1.7, 1.7]
- Logs warnings when normalization occurs
- Introduces better error handling and clearer exception messages

4. robust_controller_py27.py

- Ported for Python 2.7, to run directly on the DeepRacer
- Same functionality as robust_controller.py

5. robust_controller_recalibrated_py27.py

- Instead of clamping, this version recalibrates theta values by mapping the full range ±3.2 to ±1.7
- When tested on the DeepRacer, it receives actions from pFaces after a 2–3 minute delay

6. hybrid_controller.py (unsuccessful)

- Introduces an adaptive timeout strategy:
- 3-minute timeout for the initial connection
- 3-second timeout for subsequent queries
- Implements data caching and graceful fallback behavior to improve resilience
- Attempts but does not resolve the 2–3 minute issue

7. simple_optitrack_test.py (unsuccessful)

Used to diagnose where the delay originates. Tests:

- LocalizationServerInterface
- httplib2 with default timeout (used in RESTApiClient)
- httplib2 with custom socket-level timeout

Latest Development: Server Connection Issue

1. Simple Test vs. Full Controller:
    - The simple test version (based on synth_and_test) returned action controls from pFaces instantly
    - The full controller (robust_controller_py27) needed 2-3 minutes to establish the initial connection

2. Potential Causes:
    - Connection Reuse: The full controller may be reusing connections improperly
    - Network Stack Differences: Different methods of making HTTP requests
    - Error Handling Overhead: More complex error handling adding delays
    - Python 2.7 Limitations: Older networking libraries with less efficient implementations. Python 2.7 vs. Python 3.8:
        - Python 2.7's networking stack is outdated and less efficient
        - Python 3.8 offers significant improvements in HTTP connection handling:
            1. Modern SSL/TLS implementation with better performance
            2. More efficient socket management
            3. Better timeout handling and connection pooling
            4. Access to newer libraries like `requests` instead of `httplib2`

Tests designed to diagnose OptiTrack server connection timeout:

Each test builds on previous one to isolate specific issue causing the 3-minute delay:

1. Basic Connection Test: Tests direct HTTP connection to OptiTrack
2. RESTApiClient Test: Tests if the abstraction layer adds overhead
3. LocalizationServerInterface Test: Tests if the higher-level interface causes delays
4. pFaces Connection Test: Tests if the delay is with pFaces specifically
5. Synthesize Controller Test: Tests the controller synthesis step
6. Connection Reuse Test: Tests if reusing connections causes increasing delays
7. Full Controller Sequence Test: Breaks down the entire process with timing for each step

What to Look For

- If test6_connection_reuse.py shows increasing delays with each reused connection, that suggests connection reuse is the issue
- If test2_rest_client.py is much slower than the basic connection test, that points to the RESTApiClient implementation
- If specific steps in test7_full_controller_sequence.py add significant overhead, that indicates where error handling might be causing delays
- If all tests show consistent delays that don't appear in Python 3.8, that confirms Python 2.7 limitations

Next Steps:

1. test out the seven diagnostic tests above
2. if diagnostic unsuccessful, install Python 3.8