DeepRacer Demo Procedure

LAST UPDATED May 11, 2025

by Jackie Mara jama9909@colorado.edu
and Felipe Jimenez felipe.galarzajimenez@colorado.edu

Jump to Section:

1. Quick Reference
2. Links
3. DeepRacer Demo - Procedure
4. DeepRacer Demo - Troubleshoot
    a. previous info
    b. ventuz
    c. controller pc graphics card
5. Closeedloop_rt.py - Troubleshoot Summary & Next Steps
6. startup.py - wrapper for running closedloop_rt startup scripts

---

Quick Reference:

Media Server:

192.168.1.194:12345/OptiTrackRestServer

pFaces Server:

192.168.1.144:12345/pFaces/REST/dictionary/DeepRacer1

pFaces:

D:/Workspace/pFaces-SymbolicControl/ex_gb_fp/deepracer_rt/run_d_1_2_fast.bat

---

Links:

Google doc for this file: DeepRacer Demo Procedure
Other Github Resources https://github.com/HyConSys/CUBLab https://github.com/HyConSys/deepracer-utils

---

DeepRacer Demo - Procedure

coming soon :)

DeepRacer Demo - Troubleshoot

Ventuz - Multi-Projector Setup

Problem: When trying to configure Ventuz to span across multiple projectors as a single large display, you may encounter an issue where only one display (DISPLAY1) is visible in the Ventuz configuration app, even though all projectors show signal outside of Ventuz.

Solution: (for Ventuz 6.9)

1. Clean Graphics Drivers:
    - Use AMD Cleanup Utility to remove existing graphics drivers
    - Reinstall the latest drivers for your AMD graphics card

2. Configure AMD Eyefinity:
    - Open AMD Radeon Software
    - Navigate to the Display settings
    - Set up Eyefinity to merge multiple displays into a single large display

- This creates a "spanned" display that Ventuz can recognize as a single output

3. Ventuz Configuration:
    - After setting up Eyefinity, launch Ventuz
    - The merged display should now appear as a single output option
    - Configure this output in Ventuz for your multi-projector setup

For Ventuz 7+ (in case of future upgrades):

1. Windows 10:
    - Ventuz 7+ can handle multiple outputs without requiring display spanning
    - Add the AMD device to the Output panel 4 times (once for each output)
2. Windows 11:
    - Due to Windows changes to Full-Screen Optimization, you may still need to use AMD Eyefinity/Mosaic mode
    - This helps avoid sync issues between displays

Controller Graphics Card - HP Z8 G4 Workstation

System Information

- Model: HP Z8 G4 RCTO Base Model Workstation
- Property Code: UCB 215914 UCPC
- Serial Number: MXL12446V8

Problem 1: No Display Signal

Symptoms

- No display output
- May receive hardware error 3.3 (no graphic initialization)

Potential Causes

- Interference from Dual Port Thunderbolt 3 PCIe AiC card
- NVIDIA graphics card connection issues

Troubleshooting Steps (following official guide HP Thunderbolt 3 PCIe Card Installation)

1. Disconnect the Dual Port Thunderbolt 3 PCIe AiC card
2. Ensure at least one NVIDIA card is properly connected
3. Power on the system
4. Once display is working, update BIOS to v.2.94 Rev A (for Win 10 x64) as recommended in the Dual Port Thunderbolt 3 PCIe AiC installation guide

Problem 2: RAM Memory Recognition (Error 3.2)

Symptoms

- Error 3.2 (no memory initialization)
- System fails to recognize installed RAM

Resolution: CMOS Reset Procedure

1. Power down the system completely:
    - Shut down the system
    - Disconnect the AC power cord
    - Remove all external devices
2. Reset the CMOS:
    - Locate the yellow CMOS reset switch on the system board
    - Press and hold for 5-8 seconds
3. Establish minimal configuration:
    - Keep only essential components:
        - System board
        - Power supply

    - Processor
    - One known working memory module
4. Restore connections:
    - Reattach the access panel
    - Reconnect the power cord
5. Test the system:
    - Power on the workstation
    - Verify the system completes POST (Power On Self Test)

Closeedloop_rt.py - Troubleshoot Summary & Next Steps

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. closedloop_rt_robust.py

Added Theta Normalization:

- normalize_theta function clamps values to [-1.7, 1.7] range
- Sends warning message when theta values are being normalized

Improved Error Handling:

- Specific exception handling with detailed error messages
- Explicit mention of theta range issues in error messages
- Improved logging to help diagnose failures

2. robust_controller_py27.py

Fixed Hyperrectangle Format

Improved Connection Management:

- Better handling of HTTP connections
- Explicit error handling for connection timeouts
- *Fallback to hardcoded values when connections fail (should be removed)*

3. robust_controller_recalibrated_py27.py

Theta Value Recalibration (instead of clamping): Mapped the full range [-3.2, 3.2] to [-1.7, 1.7] to reflect accurate information from the original theta values.

4. hybrid_controller_py27.py

Adaptive Connection Strategy:

- Uses a 3-minute timeout for the initial connection to OptiTrack
- Uses a shorter 3-second timeout for subsequent requests
- Implements data caching to avoid unnecessary requests

Better Error Recovery:

- Graceful fallback to hardcoded values
- Clear error messages for different failure scenarios
- Proper cleanup of connections to avoid resource leaks

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