Implementation Roadmap for Random Gait MPPI

Current System Overview

1. Gait Generation (`PeriodicGaitGenerator`)

- Fixed gait patterns (trot, pace, etc.)
- Predetermined phase offsets
- Fixed duty factors
- No runtime optimization

2. Foothold Planning (`FootholdReferenceGenerator`)

- Determines foot placement locations
- Based on predefined gait patterns
- Uses kinematic and stability constraints
- No dynamic optimization

3. MPPI Controller (`centroidal_nmpc_jax.py`)

- Optimizes Ground Reaction Forces only
- Uses predetermined contact sequences
- No control over gait timing or foothold positions

Project Objectives

1. Randomize Gait Patterns

- Remove dependency on fixed gaits
- Implement variable stance/swing durations
- Maintain following constraints:
  - Minimum stance duration
  - Support polygon stability
  - Leg coordination rules

2. MPPI-based Optimization

- Stage 1: Timing Optimization

  - When to transition between stance/swing
  - Duration of each phase
  - Stability-aware sampling

- Stage 2: Position Optimization

  - Where to place feet during touchdown
  - Kinematic reachability
  - Terrain adaptation

Implementation Strategy

Phase 1: Random Gait Generator

1. Create new class `RandomGaitGenerator`

    - Implement basic random gait sampling
    - Add stability constraints:
        - Minimum support legs (≥2)
        - Adjacent leg coordination
        - Maximum swing duration
    - Unit tests for gait feasibility

2. Create adapter interface

    - Bridge between `RandomGaitGenerator` and existing `PeriodicGaitGenerator`
    - Allow switching between random/periodic gaits
    - Maintain backward compatibility

3. Define Gait Parameters

    Essential Parameters:

    - Stance/swing duration ranges

        - Minimum stance duration (e.g., 0.2-0.3s)
        - Maximum swing duration (e.g., 0.4s)
        - These are critical for physical feasibility

    - Support polygon requirements

        - Minimum number of supporting legs (≥2)
        - Adjacent leg coordination rules
        - These ensure basic stability

    Optional Parameters:

    - Energy efficiency metrics

        - Can be added later when optimizing performance
        - Not critical for basic gait generation

    - Transition costs between states

        - Useful for smoothing transitions
        - Can be implemented in later iterations

Phase 2: MPPI Integration

1. Extend MPPI sampling space

    - Add gait parameters to optimization
    - Modify cost function to include gait costs

- Keep GRF optimization intact

2. Create new MPPI wrapper

   - Separate random gait MPPI from existing implementation
   - Use inheritance to extend `Sampling_MPC`
   - Add new configuration options

3. Cost Function Design

   - Primary Costs (Timing Optimization)
     - Phase transition feasibility
     - Minimum stance duration enforcement
     - Support polygon stability
     - Leg coordination penalties
   - Stability Costs
     - Center of mass trajectory tracking
     - Support polygon metrics
     - Angular momentum bounds
   - Secondary Costs (Optional)
     - Gait symmetry
     - Transition smoothness
     - Movement efficiency

Phase 3: Foothold Optimization

1. Create `FootholdOptimizer` class

   - Sample foothold positions
   - Add kinematic constraints
   - Consider terrain information

2. Integration with random gaits

   - Coordinate foothold and gait optimization
   - Add combined cost functions
   - Maintain stability guarantees

3. Performance Optimization

   - JAX implementation for parallel sampling
   - Efficient contact sequence representation
   - Warm starting from previous solutions

Phase 4: Testing & Validation

1. Unit Tests

   - Gait feasibility tests
   - Foothold reachability tests
   - Stability constraint tests

2. Integration Tests
    - Combined gait-foothold optimization
    - Performance benchmarks
    - Compare with periodic gaits
3. Simulation Testing
    - Test in different scenarios
    - Measure success metrics
    - Debug and optimize parameters

Directory Structure

```
quadruped_pympc/
├── helpers/
│   ├── random_gait_generator.py     # New file
│   ├── gait_adapter.py             # New file
│   └── foothold_optimizer.py       # New file
├── controllers/
│   └── sampling/
│       └── random_gait_mppi.py     # New file
└── tests/
    ├── test_random_gait.py
    ├── test_foothold_opt.py
    └── test_combined_mppi.py
```

Implementation Order

1. Basic `RandomGaitGenerator` with tests
    - Basic gait mask generation
    - Stability constraints
    - Unit tests for constraints
2. Adapter interface implementation
    - Bridge to existing `PeriodicGaitGenerator`
    - Contact sequence conversion
3. MPPI extension for gait optimization
    - Timing parameter sampling
    - Cost function implementation
    - Testing with fixed footholds
4. Foothold optimization integration
5. Full system integration
6. Testing and validation
7. Parameter tuning and optimization

Success Metrics

- Gait feasibility rate

- Stability measures
- Tracking performance
- Computational efficiency/Computation time per optimization cycle