Robust Quadruped RL - Complete To-Do List

📋 Phase 1: Foundation (Week 1-2)

1.1 Environment Setup ✅

- [ ] Install MuJoCo
- [ ] Install gymnasium, stable-baselines3
- [ ] Set up project structure
- [ ] Create virtual environment
- [ ] Test basic imports work

1.2 Get RealAnt Working

- [ ] Find/install RealAnt environment
  - Check: https://github.com/AaltoVision/realant-rl
  - Or create basic wrapper for MuJoCo ant
- [ ] Test environment loads: `env = gym.make('RealAnt-v0')`
- [ ] Understand observation space (28 dimensions)
- [ ] Understand action space (8 dimensions)
- [ ] Record video of random policy

1.3 Define Success Metrics

- [ ] Target velocity: 0.5-1.0 m/s
- [ ] Success: maintain velocity for 5 seconds
- [ ] Evaluation episodes: 100
- [ ] Create evaluation script that measures:
  - Average velocity
  - Distance traveled
  - Success rate
  - Episode length

📋 Phase 2: Baseline PPO (Week 2-3)

2.1 Implement Basic PPO Training

```python
# Goal: Robot walks forward at 0.5 m/s for 5 seconds
```

- [ ] Create `train_ppo_baseline.py`
- [ ] Use stable-baselines3 PPO
- [ ] Design reward function:
  - Forward velocity reward
  - Alive bonus
  - Control cost penalty
- [ ] Train for 1M steps initially
- [ ] Log to TensorBoard
- [ ] Save best model

2.2 Tune Hyperparameters

- [ ] Learning rate: try [1e-4, 3e-4, 1e-3]
- [ ] Batch size: try [64, 256, 2048]
- [ ] Network size: try [64,64], [128,128], [256,256]
- [ ] Find best configuration
- [ ] Document what works

2.3 Evaluate Baseline

- [ ] Run 100 evaluation episodes
- [ ] Record metrics:
  - Success rate: ___% (target: >90%)
  - Avg velocity: ___ m/s
  - Avg distance: ___ m
- [ ] Save videos of best episodes
- [ ] Create baseline_results.json

📋 Phase 3: PPO + SR²L (Week 3-4)

3.1 Understand SR²L

- [ ] Read SR²L paper/theory
- [ ] Understand smooth regularization concept
- [ ] Plan implementation approach

3.2 Implement SR²L

- [ ] Create `sr2l_ppo.py`
- [ ] Add smoothness loss to PPO:

```
python
```

```
# Pseudocode
perturbed_obs = obs + small_noise
smooth_loss = MSE(policy(obs), policy(perturbed_obs))
total_loss = ppo_loss + lambda * smooth_loss
```

- [ ] Add SR²L coefficient (λ = 0.01)
- [ ] Verify loss is computed correctly

3.3 Train PPO + SR²L

- [ ] Create `train_ppo_sr2l.py`
- [ ] Use same hyperparameters as baseline
- [ ] Train for same number of steps
- [ ] Monitor smooth_loss in logs

3.4 Evaluate PPO + SR²L

- [ ] Run evaluation (clean environment)
- [ ] Compare to baseline:
  - Success rate: ___% (should be similar)
  - Motion smoothness (new metric)
- [ ] Verify actions are smoother

📋 Phase 4: Domain Randomization Setup (Week 4-5)

4.1 Create Fault Injection Wrapper

- [ ] Create `fault_injection_wrapper.py`
- [ ] Start simple: single joint dropout
- [ ] Implement step-by-step:

```
python

# Step 1: Fixed joint failure
# Step 2: Random joint selection
# Step 3: Multiple joints
# Step 4: Probability-based
```

4.2 Test Fault Injection

- [ ] Create `test_faults.py`
- [ ] Verify joints actually stop working
- [ ] Test different fault modes:

- Zero torque (motor off)
- Joint locking (maintain position)
- Weak motor (reduced torque)

- [ ] Visualize robot with faults

4.3 Create Curriculum Manager

- [ ] Create `curriculum.py`
- [ ] Implement 3 phases:
  1. No faults (0-200k steps)
  2. Single faults (200k-600k steps)
  3. Multiple faults (600k+ steps)
- [ ] Test phase transitions work

📋 Phase 5: PPO + Domain Randomization (Week 5-6)

5.1 Integrate DR into Training

- [ ] Create `train_ppo_dr.py`
- [ ] Wrap environment with fault injection
- [ ] Use curriculum manager
- [ ] Log fault statistics

5.2 Train with Actuator Faults Only

- [ ] Start with single joint failures
- [ ] Probability: 20% per episode
- [ ] Train for 5M steps
- [ ] Monitor:
  - Success rate over time
  - Performance per fault type

5.3 Add Multiple Joint Failures

- [ ] Enable 2-3 joint failures
- [ ] Increase fault probability to 40%
- [ ] Continue training
- [ ] Track recovery strategies

5.4 Evaluate PPO + DR

- [ ] Test on clean environment
- [ ] Test with single faults
- [ ] Test with multiple faults
- [ ] Compare to baseline

📋 Phase 6: Sensor Noise (Week 6)

6.1 Implement Sensor Noise

- [ ] Add to fault wrapper:
  - Position sensor noise (σ=0.05)
  - Velocity sensor noise (σ=0.1)
  - Orientation noise (σ=0.02)
- [ ] Test noise is applied correctly

6.2 Train with Sensor Noise Only

- [ ] Create variant with just noise
- [ ] No actuator faults yet
- [ ] Verify robot still learns

6.3 Combine Faults + Noise

- [ ] Enable both actuator faults AND sensor noise
- [ ] Use full curriculum
- [ ] Train complete system

📋 Phase 7: Full Method (Week 7-8)

7.1 PPO + DR + SR²L

- [ ] Create `train_ppo_dr_sr2l.py`
- [ ] Combine all components:
  - PPO base algorithm
  - SR²L smoothness
  - Domain randomization
  - Curriculum learning
- [ ] Train for 10M steps

7.2 Final Training Runs

- [ ] Run all 4 ablations:

1. PPO only
2. PPO + SR²L
3. PPO + DR
4. PPO + DR + SR²L

- [ ] Use same seeds for fairness
- [ ] Save all models

📋 Phase 8: Evaluation & Analysis (Week 8-9)

8.1 Comprehensive Evaluation

- [ ] Create `evaluate_all.py`
- [ ] Test scenarios:
  1. Clean (no faults)
  2. Single joint failure
  3. Multiple joint failures
  4. Sensor noise only
  5. Combined faults + noise
- [ ] 100 episodes each

8.2 Statistical Analysis

- [ ] Compute mean ± std for all metrics
- [ ] Run statistical tests (t-test)
- [ ] Create results tables
- [ ] Generate plots:
  - Learning curves
  - Success rates by condition
  - Ablation comparison

8.3 Failure Analysis

- [ ] Identify failure modes
- [ ] Analyze recovery strategies
- [ ] Create failure taxonomy
- [ ] Document interesting behaviors

📋 Phase 9: Visualization & Documentation (Week 9-10)

9.1 Create Visualizations

- [ ] Training curve comparisons
- [ ] Bar charts of success rates
- [ ] Videos of each method
- [ ] Failure recovery examples

9.2 Write Up Results

- [ ] Document all hyperparameters
- [ ] Create results tables
- [ ] Write analysis
- [ ] Prepare for final report

🎯 Success Milestones

Milestone 1: Basic Walking ✓

- PPO baseline achieves 90%+ success rate
- Robot walks at 0.5+ m/s consistently

Milestone 2: Smooth Walking ✓

- SR²L version has visibly smoother motion
- Maintains baseline performance

Milestone 3: Single Fault Robustness ✓

- 70%+ success with one failed joint
- Clear recovery behaviors

Milestone 4: Multi-Fault Robustness ✓

- 45%+ success with 2-3 failed joints
- Degrades gracefully

Milestone 5: Full Robustness ✓

- Best performance from combined method
- Statistical significance achieved

📊 Tracking Template

For each experiment, track:

```
Experiment: PPO_baseline
Date: 2025-07-XX
Training time: XX hours
Final success rate (clean): XX%
Final success rate (1 fault): XX%
Final success rate (2+ faults): XX%
Average episode reward: XX
Best model checkpoint: experiments/ppo_baseline/model_5M.pt
Notes: [Any observations]
```

🚦 Go/No-Go Checkpoints

Before moving to next phase, ensure:

- [ ] Current phase success criteria met
- [ ] Code is committed to git
- [ ] Results are logged and saved
- [ ] Videos demonstrate behavior
- [ ] Ready to add complexity

💡 Pro Tips

1. **Start simple**: Get basic walking first
2. **One change at a time**: Don't add SR²L and DR together initially
3. **Extensive logging**: Log everything, you'll need it later
4. **Visual debugging**: Record videos often
5. **Patience**: Full training takes time (24-48 hours per run)

This systematic approach ensures you understand each component's contribution and can debug issues effectively!