Pre-requisites

- URDF MODEL
- TRAIN ENV
- DEPLOY ENV
- STATE TRACKER

ANYBIPE

Requirements

◆ **Compulsory**

Env description, task description, estimator, reward mapping table, safety criterion

◆ **Optional**

Reference reward function, reference policy from previous work (teacher), additional prompts, custom observation

Module 1

Prompts
- ✓ *Coding Tips*
- ✓ *Regulations*
- ✓ *Feedback*

Env + Task Description

References Custom Configs

LLM

Reward Functions

```
class AnybipeRewards():
    def __init__(self, env):
        self.env = env

    def load_env(self, env):
        self.env = env

    def _tracking_lin_vel(self):
        # Tracking of linear velocity commands (xy axes)
        lin_vel_error = torch.sum(torch.square(self.env.commands[:, :2] - self.env.base_lin_
        return torch.exp(-lin_vel_error / self.env.cfg.rewards.tracking_sigma)

    def _tracking_ang_vel(self):
        # Tracking of angular velocity commands (yaw)
        ang_vel_error = torch.square(self.env.commands[:, 2] - self.env.base_ang_vel[:, 2])
        return torch.exp(-ang_vel_error / self.env.cfg.rewards.tracking_sigma)
```

Feedback Compiler

We deployed the trained RL policy to our robot and evaluated its performance in reality. We converted the reward functions that can be calculated by real-world sensors and actuators
…
here are the results of the evaluation:
…

Module 2

Previous Work

Reference Policy

Supervised

Module 3

Best Model & Eval Feedback

Reality Eval

Pass Safety Check

Gazebo Eval

Estimator + **Observation**

Top Policies & Top Rewards

$\mathcal{F}$

Homomorphic Evaluation Estimator

Module Function

Module 1: LLM Reward Function Generation

Module 2: Teacher Guided PPO Training

Module 3: Deploy, Evaluation, and Feedback