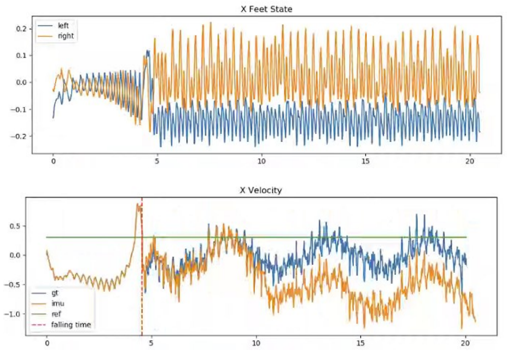
```python
def _reward_action_rate(self):
    # Penalize changes in actions
    return torch.sum(torch.square(self.env.last_actions - self.env.actions), dim=1)

def _reward_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_vel[:, :2]), dim=1)
    return torch.exp(-lin_vel_error / self.env.cfg.rewards.tracking_sigma)

def _reward_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)
```

Original Reward



State Estimator + Convert Script

```python
for term in env_terms:
    term_name = term.split(".")[-1]
    if "shape" in term:
        term_name = term.split(".")[-2]
    # check if term is an array slicing
    slicing = False
    if "[" in term:
        term_name = term_name.split("[")[0]
        slicing = True

    if term_name not in func_map.keys():
        # unconvertable
        return None
    new_term = term.replace(term_name, func_map[term_name])
    if slicing:
        new_term = replace_first_dim_with_i(new_term)
    lines[i] = lines[i].replace(term, new_term)
```

```python
func_map = {
    "base_lin_vel": "base_lin_vel",
    "base_ang_vel": "base_ang_vel",
    "torques": "torques[idx:idx+1]",
    "dof_vel": "dq[idx:idx+1]",
    "last_dof_vel": "last_dq[idx:idx+1]",
    "dof_acc": "ddq",
    "actions": "q[idx:idx+1]",
    "dt": "dt[idx:idx+1]",
    "last_actions": "last_q[idx:idx+1]",
    "survival_time": "survival_time",
    "commands": "commands",
    "torque_limits": "torque_limits",
    "feet_state": "feet_state",
}
```

Homomorphic Reward

```python
def _reward_action_rate(self, idx):
    # Penalize changes in actions
    return torch.sum(torch.square(self.env.last_q[idx:idx+1] - self.env.q[idx:idx+1]), dim=1)

def _reward_tracking_lin_vel(self, idx):
    # Tracking of linear velocity commands (xy axes)
    lin_vel_error = torch.sum(torch.square(self.env.commands[idx:idx+1,  :2] - self.env.base_lin_vel[idx:idx+1,  :2]), dim=1)
    return torch.exp(-lin_vel_error / self.reward.tracking_sigma)

def _reward_tracking_ang_vel(self, idx):
    # Tracking of angular velocity commands (yaw)
    ang_vel_error = torch.square(self.env.commands[idx:idx+1,  2] - self.env.base_ang_vel[idx:idx+1,  2])
    return torch.exp(-ang_vel_error / self.reward.tracking_sigma)
```