RoboSkate

Midterm Project Presentation

Michelle Bettendorf
Gintautas Palinauskas
Meriç Sakarya
Batuhan Yumurtacı
Finn Süberkrüb

Cloud-Based Machine Learning in Robotics
Summer Semester 2021

Agenda

- Recap of Objective

- Work done so far:
  - Current System Architecture
  - Reward Functions
  - Image Processing
  - Callback Functions
  - First Results
  - Open Issues

- Future Work

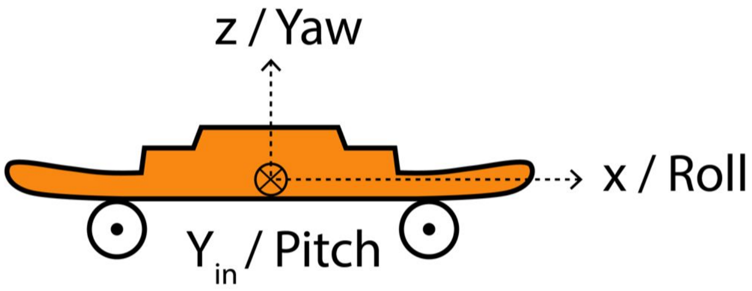
Recap of our goals

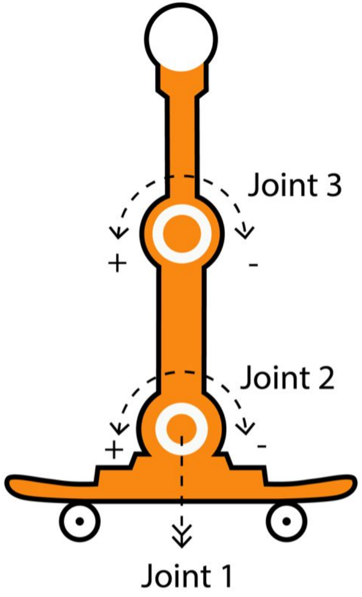
Goals:

- Setting up the simulation
- Selecting the best suited algorithm
- Teaching the agent basic movement in DummyBall
- Teaching the agent basic movement in RoboSkate
- Reaching checkpoints in game
- Finishing the whole level

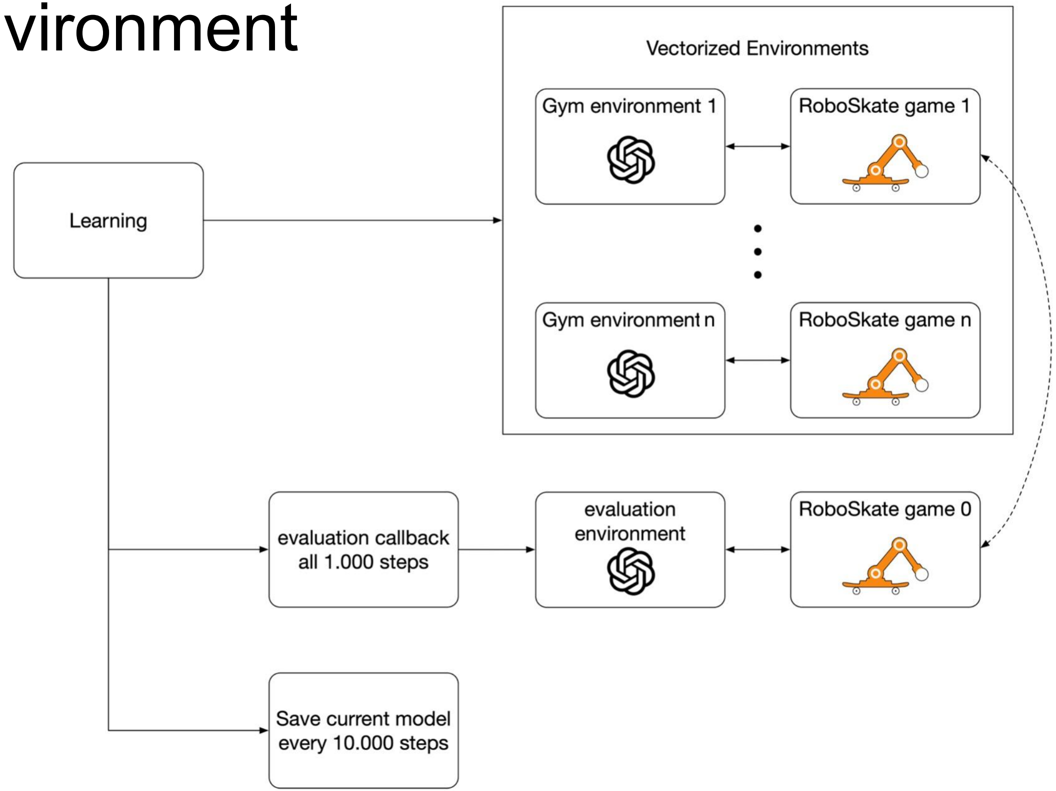
Agent

cam

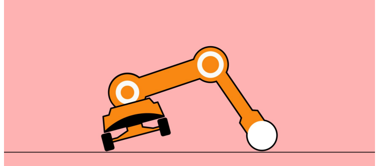




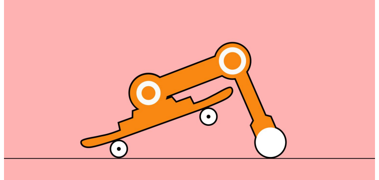
Vectorized Environment



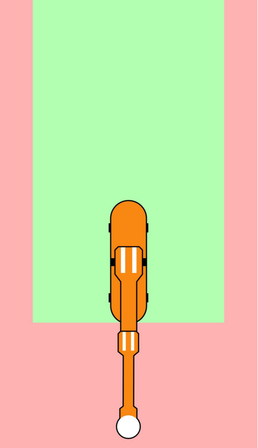
Termination



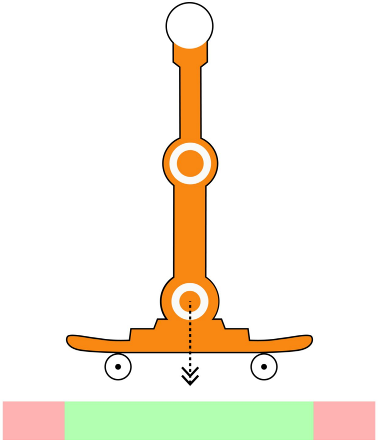
Roll angle too high



Pitch angle too high



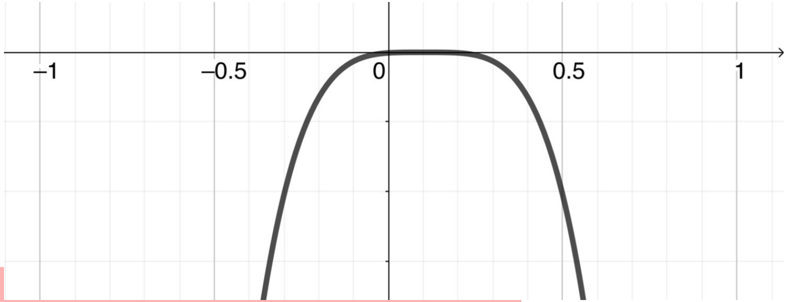
wrong direction

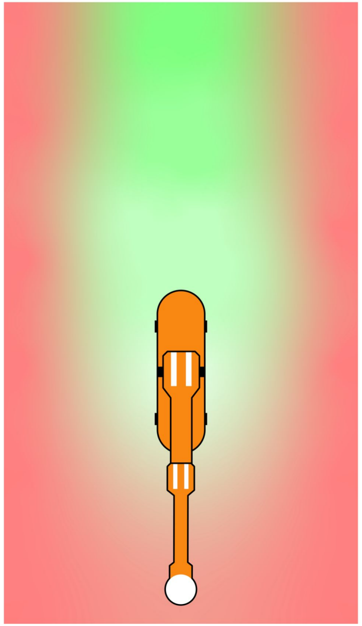


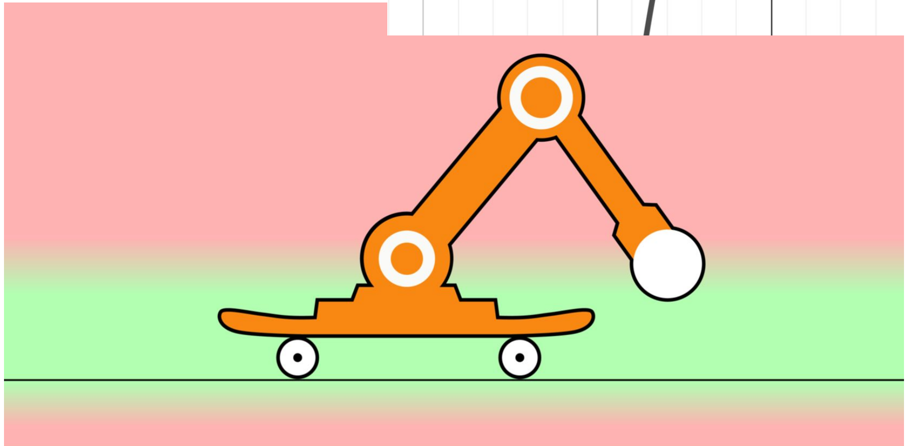
Joint velocity 1 too high

Reward function

$$\text{BallHighReward} = -\,(\text{BallHight} - 0.2)^4$$







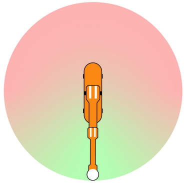
First results

- 10 environments
- 20M steps
- A2C Algorithm
- MLP Policy
- 18 Observation inputs
- Reward for:
    - travelled distance
    - low ball

https://youtu.be/PyfHF5J3TUI

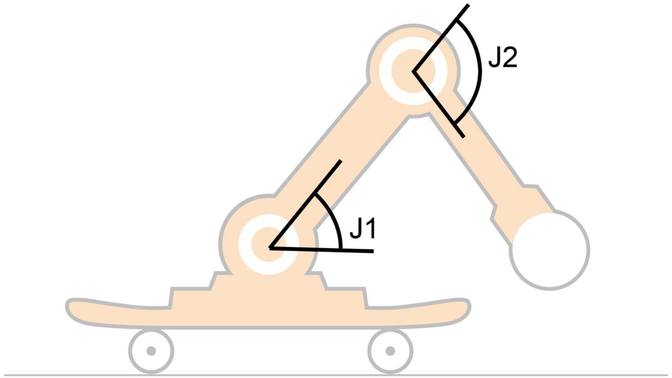
First results

- 10 environments
- 10M steps
- 28h
- A2C Algorithm
- MLP Policy
- 9 Observation inputs
- Reward for:
    - travelled distance
    - low ball
    - ball behind

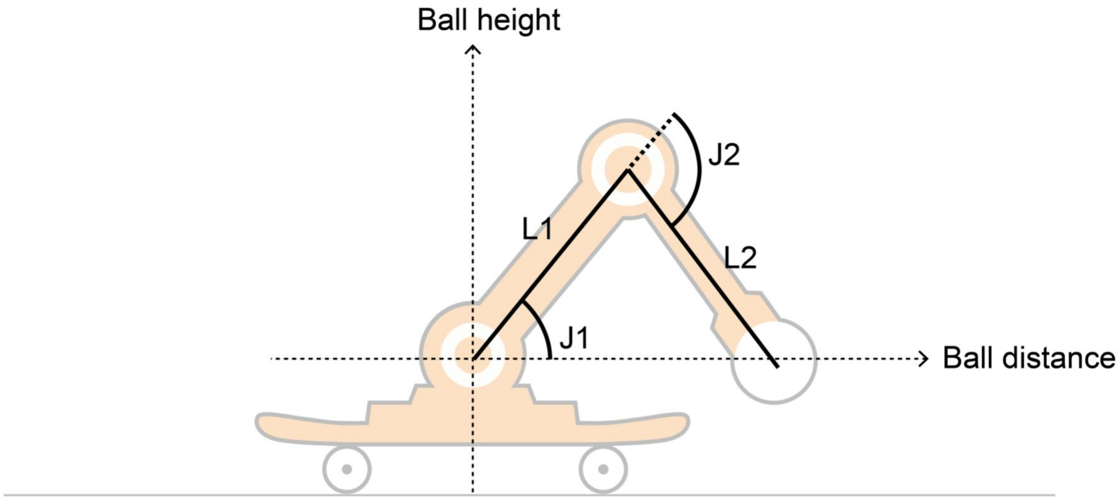


https://youtu.be/zSduDkrXD2k

2D State Space



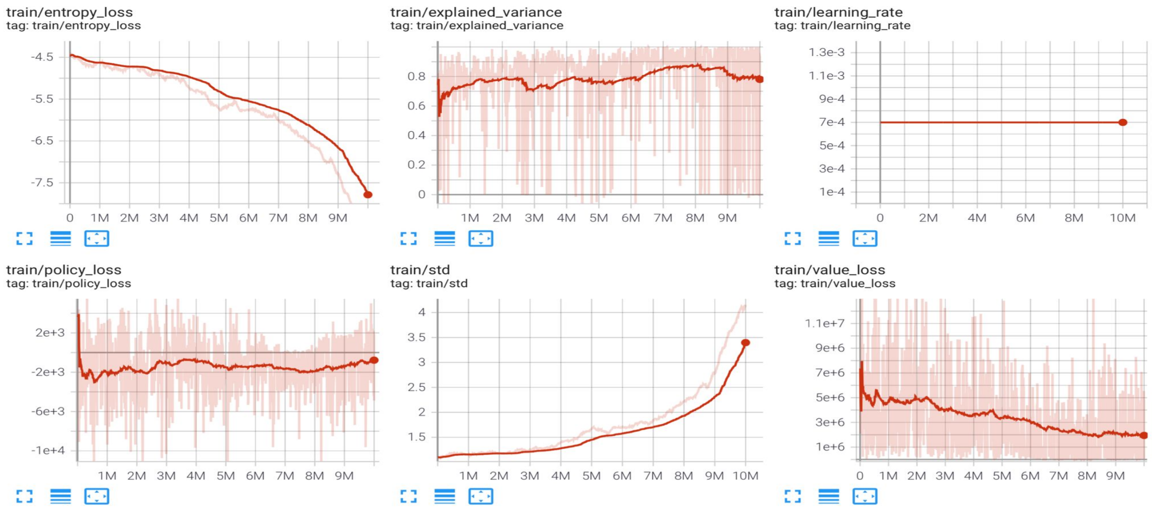
2D State Space
Cartesian coordinates



Ball height	= cos(J1 + J2) L2	+ cos(J1) L1
Ball distance	= sin(J1 + J2) L2	+ sin(J1) L2

Tensorboard Callback

- Uses its own thread.
- Example: `self.logger.record('reward/mean', np.mean(self.locals.get("rewards")))`
- Allows great visualization of the training results:

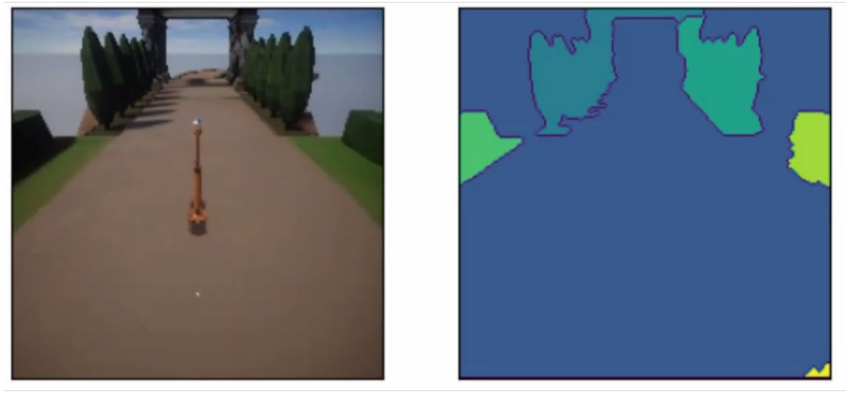


Evaluation Callback

- Evaluates the performance of the agent
- Uses separate evaluation environment
- Goal: Frequently checking up on the training to stop early
- Saves best model for further training or testing
- ```
  eval_callback = EvalCallback(eval_env,
  best_model_save_path="./scripts/python/RoboSkate/agent_models/",
  log_path="./scripts/python/RoboSkate/agent_models/",
  eval_freq=best_agent_eval_freq,
  n_eval_episodes=best_agent_n_eval_episodes,
  deterministic=True, render=False)
  ```
  ```

Image Preprocessing RoboSkate

First try: Watershed



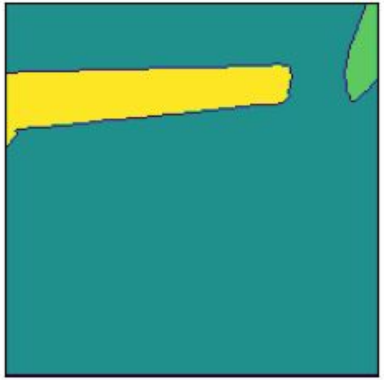


Image Preprocessing RoboSkate

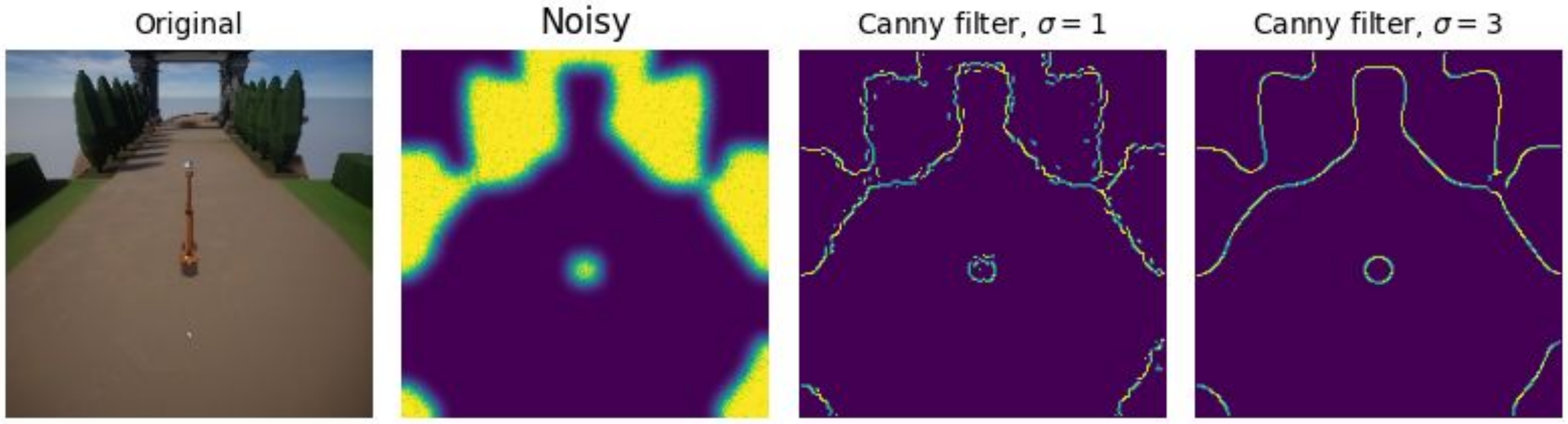


Image Preprocessing: Testing on DummyBall

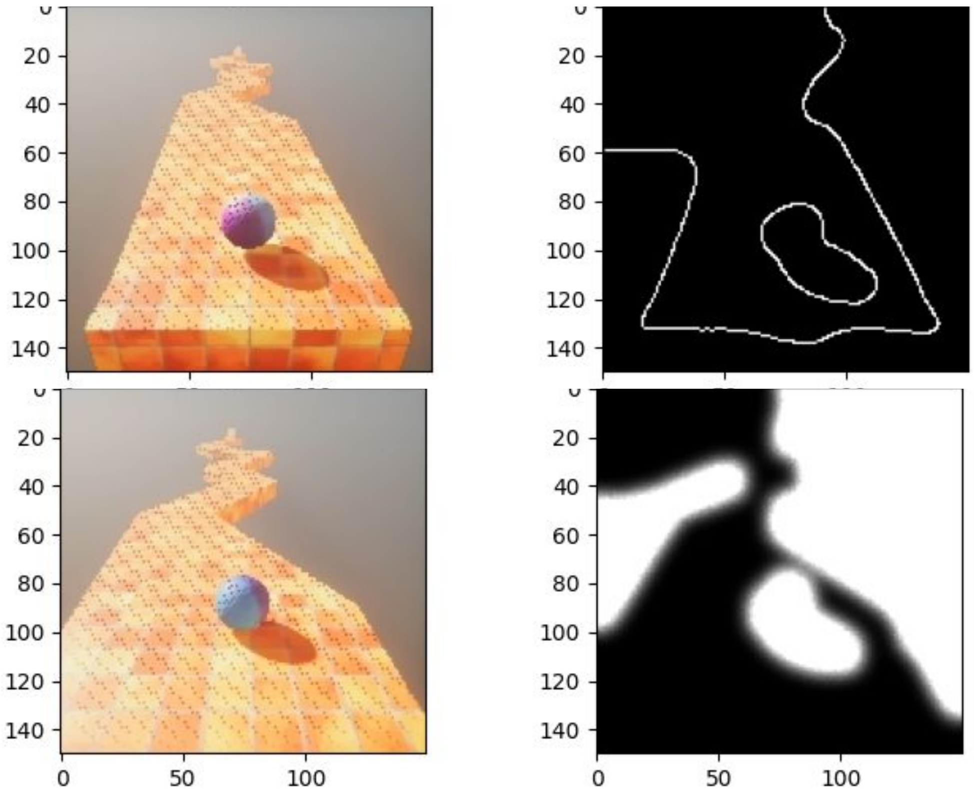
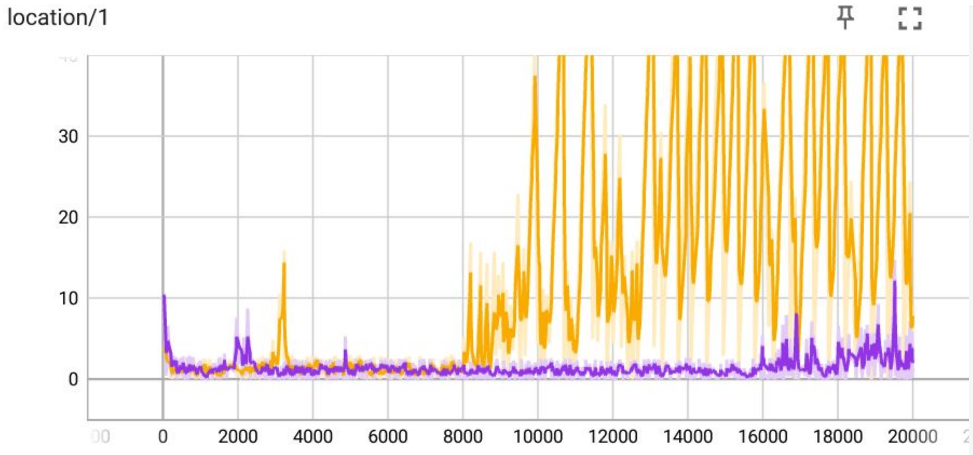
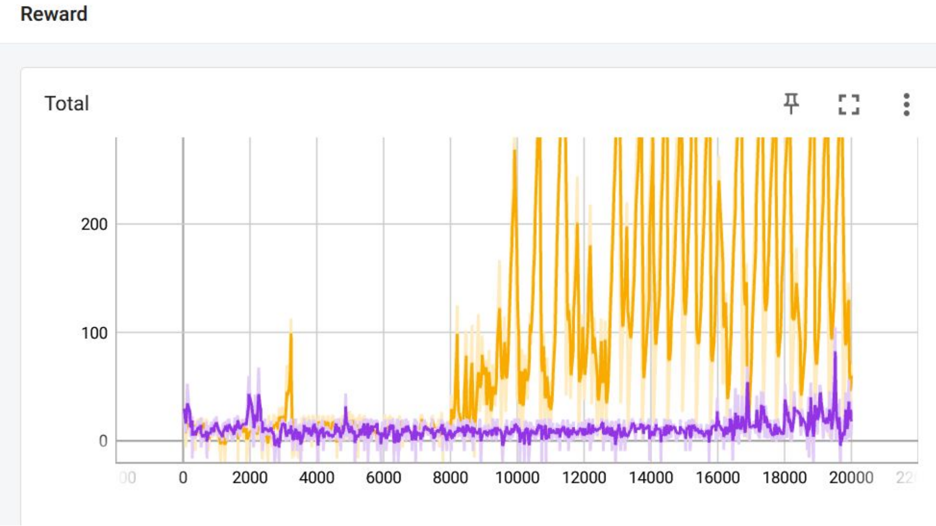


Image Preprocessing: Testing on DummyBall

A2C algorithm:





Canny filter

Noisy image

DummyBall learning via images

Conditioning the observations and actions:

- Calculate z distance from platform
- Scaling action with 50

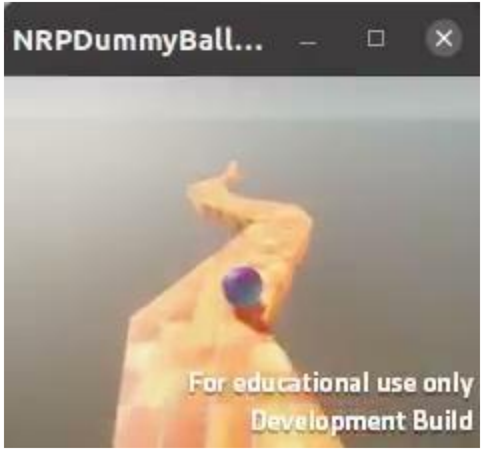
Early termination:

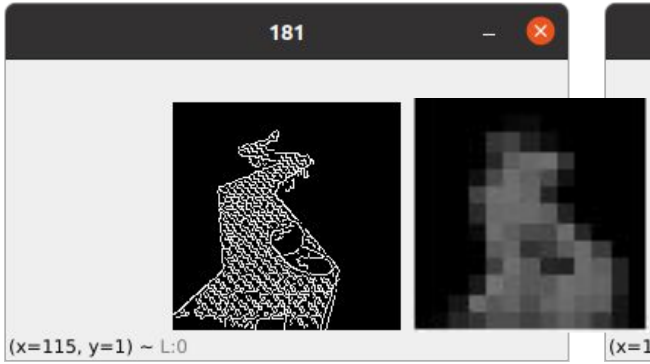
- When z distance < -1
- When x distance > 60

Image preprocessing using cv2:

1. Read 150x150 image
2. Convert to grayscale
3. Use Canny edge detector

A2C after 20000 steps





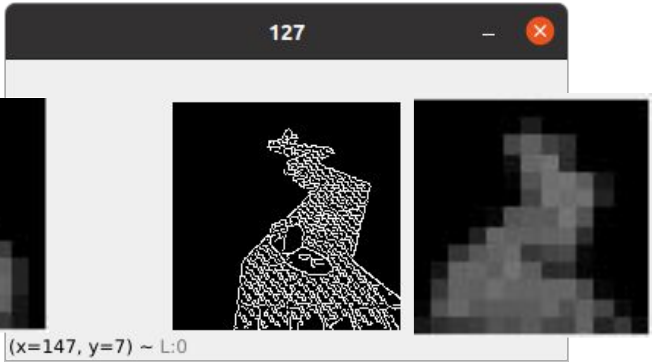
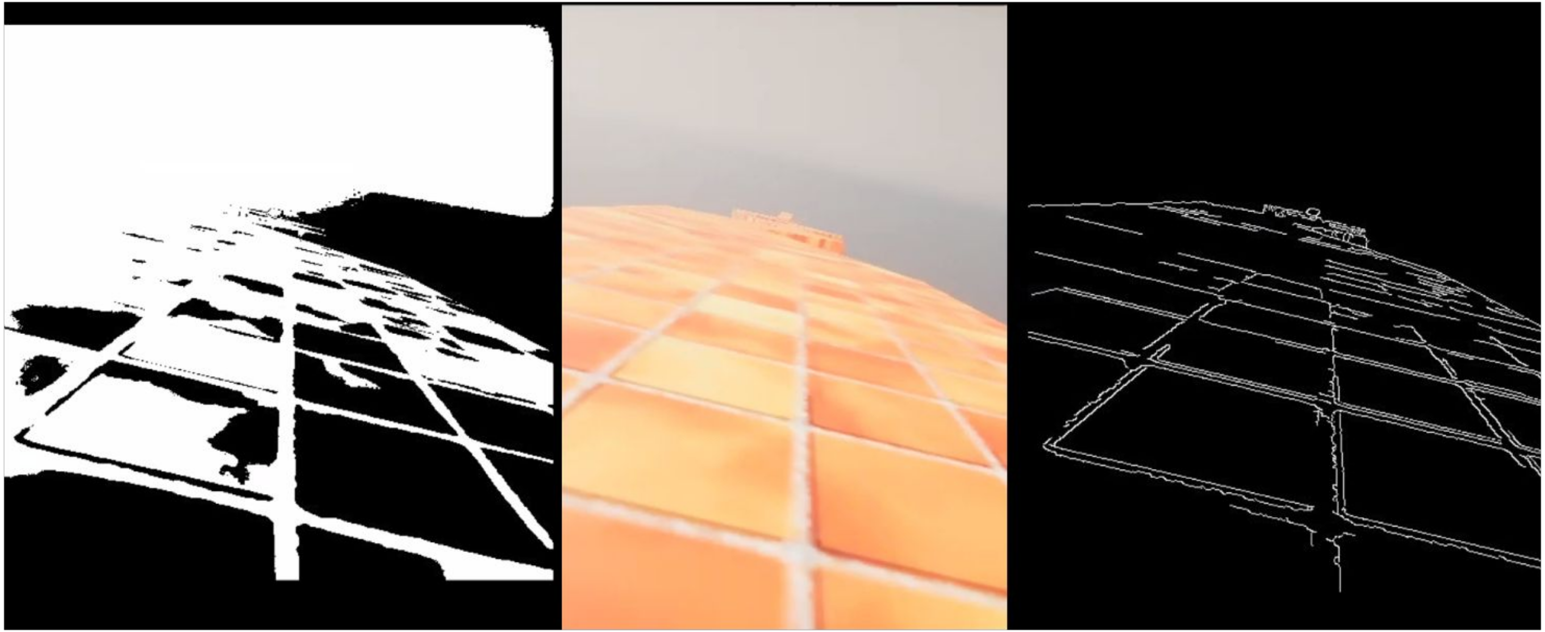


Image preprocessing ball game

Threshold method Original image Canny method



Neural network architectures

```
# Default network architecture, from stable-baselines
if net_arch is None:
    if features_extractor_class == NatureCNN:
        net_arch = []
    else:
        net_arch = [dict(pi=[64, 64], vf=[64, 64])]
```

MlpPolicy

Policy network   Value function network

```
def __init__(self, observation_space: gym.spaces.Box, features_dim: int = 512):
    super(NatureCNN, self).__init__(observation_space, features_dim)

    self.cnn = nn.Sequential(
        nn.Conv2d(n_input_channels, 32, kernel_size=8, stride=4, padding=0),
        nn.ReLU(),
        nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=0),
        nn.ReLU(),
        nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=0),
        nn.ReLU(),
        nn.Flatten(),
    )

    # Compute shape by doing one forward pass
    with th.no_grad():
        n_flatten = self.cnn(th.as_tensor(observation_space.sample()[None]).float()).shape[1]

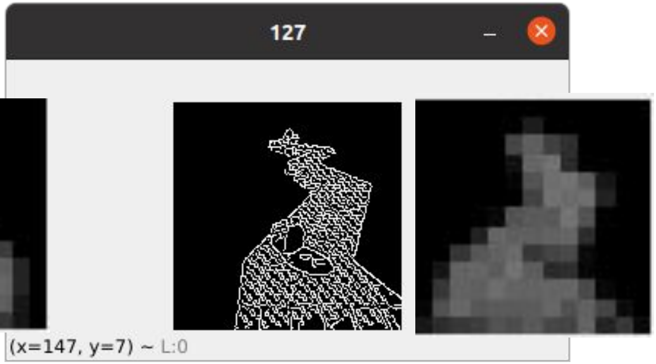
    self.linear = nn.Sequential(nn.Linear(n_flatten, features_dim), nn.ReLU())

def forward(self, observations: th.Tensor) -> th.Tensor:
    return self.linear(self.cnn(observations))
```

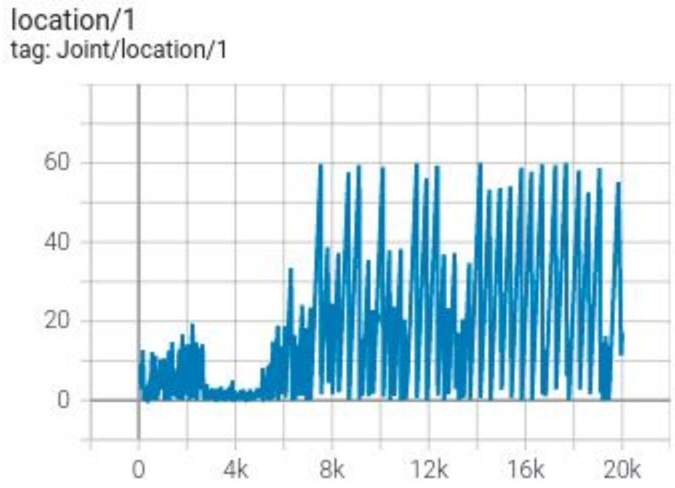
CnnPolicy

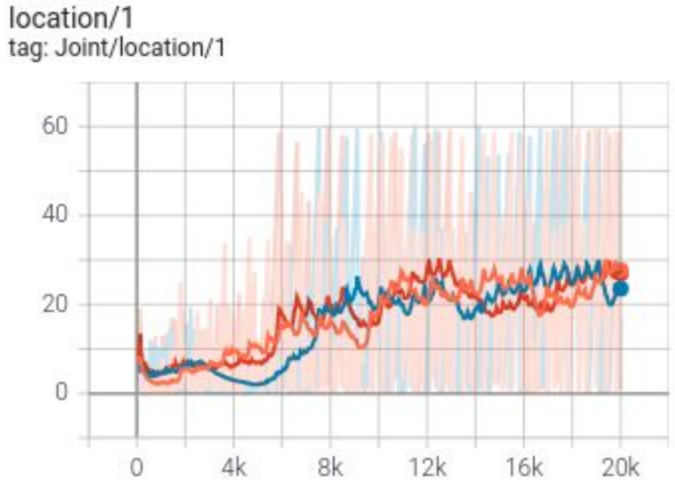
150x150 image after convolutional layers:

1. (150 - 8) / 4 + 1 = 36
2. (36 - 4) / 2 + 1 = 17
3. (17 - 3) / 1 + 1 = 14



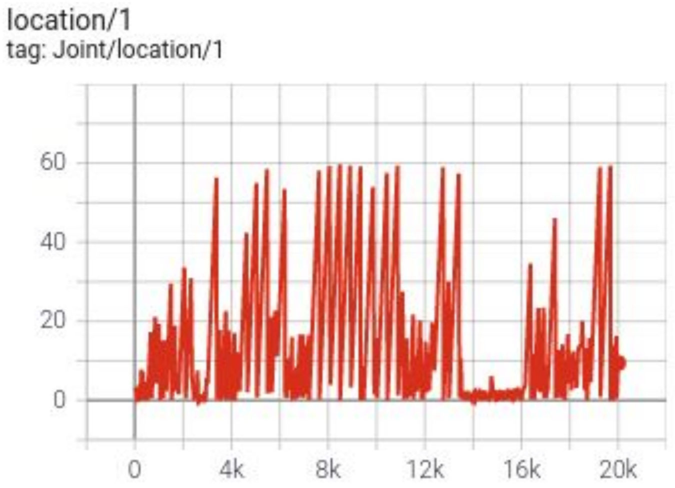
Agent learning through tensorboard (PPO)

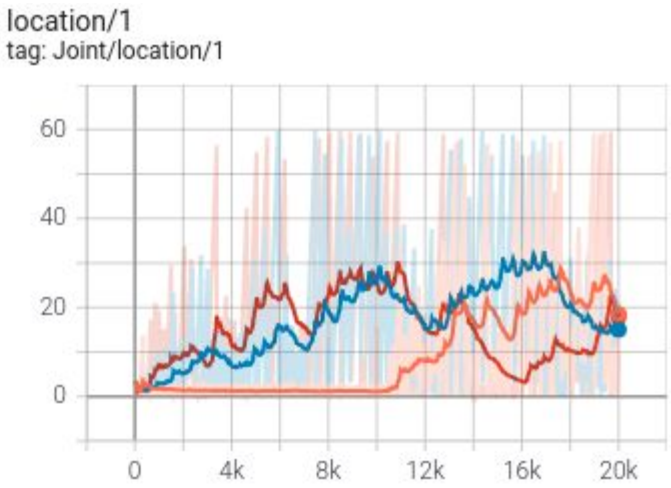




Using only default settings

Agent learning through tensorboard (A2C)





Using default settings except n_steps = 10

PPO

- Advantages:
  - Easy code
  - Sample efficient
  - Easy to tune
- On-Policy → No replay buffer
- Clipped Objective Function → Policy does not stray away

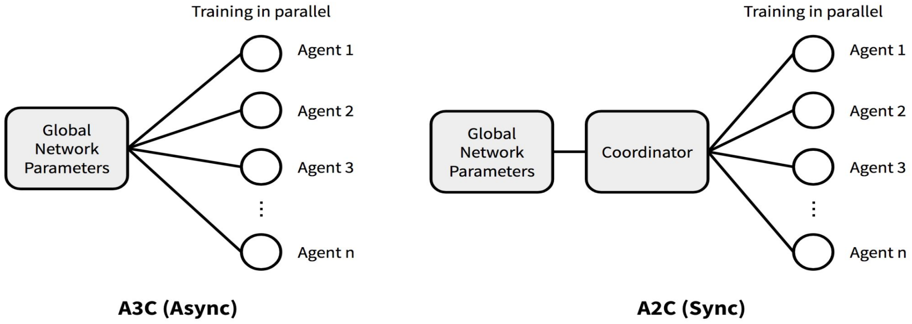
Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

PPO Algorithm [1]

A2C

- A3C
  - Asynchronous updates
  - Architectures share layers between the policy and value function
  - Policy gradient method
  - Multiple actors
  - Thread-specific agents → Policies of different versions → Update sub-optimal
- A2C: Synchronous version of Advantage Actor Critic (A3C)

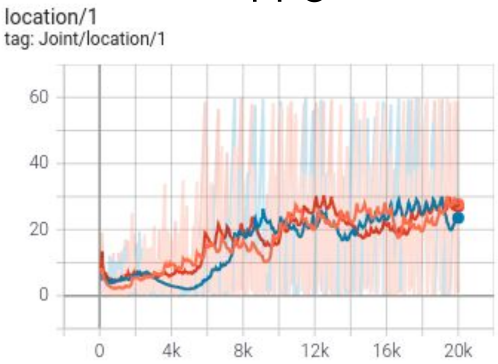


Comparison between A2C and A3C systems

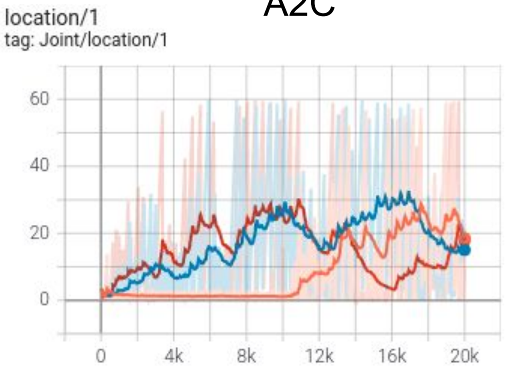
A2C vs PPO

- PPO achieved better results
- PPO converged faster
- A2C is simpler to use
- PPO is more robust to hyper-parameter changes

PPO



A2C



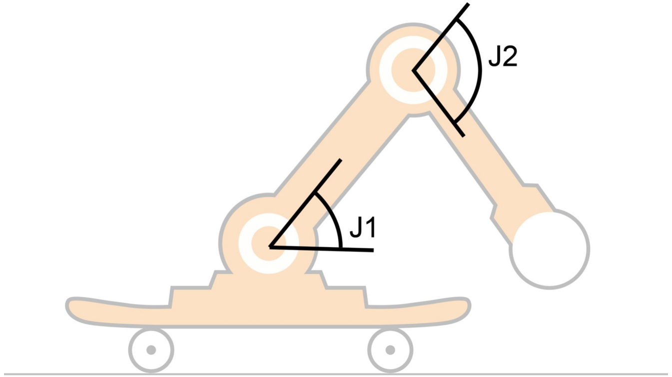
Current Issues → Possible Solutions

- Reward function too specific → Imitation Learning / Simpler environment
- NRP → Ready for DummyBall / Needs work with RoboSkate
- GPU-intensive → Simpler Environment
- Game-Physics wacky → Needs to be investigated or validated

Simple RoboSkate Environment

Create a 2D simple Roboskate:

- Faster training (lighter)
- Faster hyperparameter optimization
- Faster reward function testing
- Enough for teaching basic movement
  - Going forward/backward
  - Accelerating
  - Brake



Reward Function

- Needs tailoring
- Indeterminate functions are not performing well
- Specific instructions needed
- Sparsity is a huge problem → Imitation learning

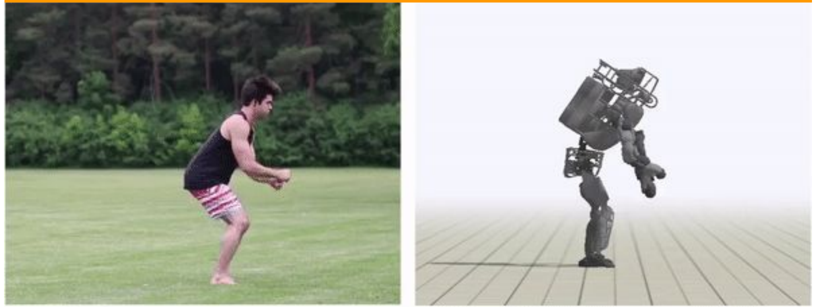
Imitation Learning

Feature and reward engineering is not always straightforward.

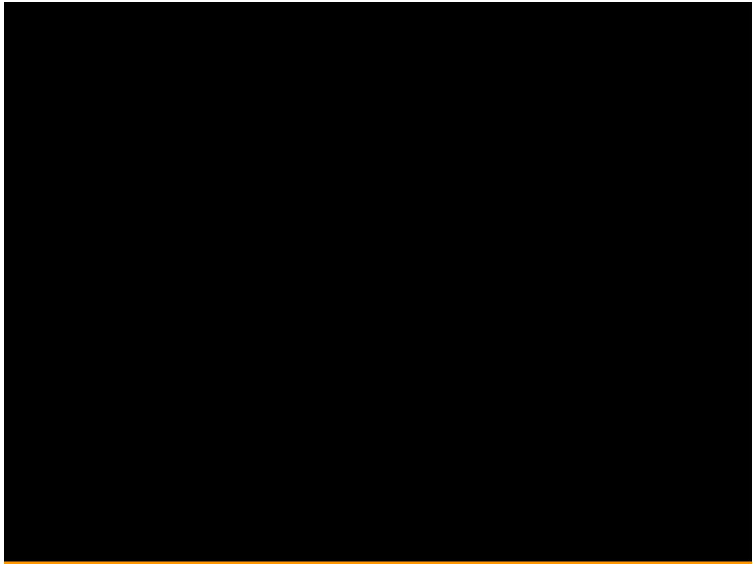
Hard-coded agent

Human player

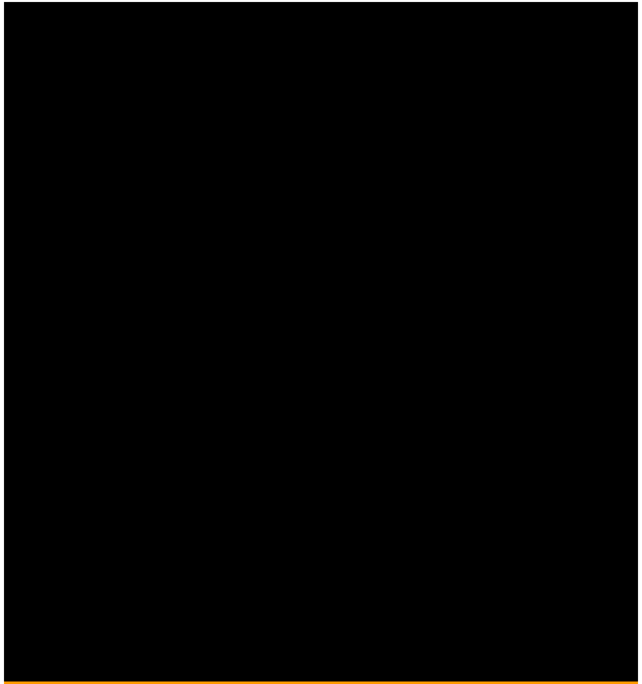
Learn to perform a task from demonstrations.



Imitation Learning



Hard-coded agent



Teleoperation demonstration
(human player)

Generative Adversarial Imitation Learning (GAIL)

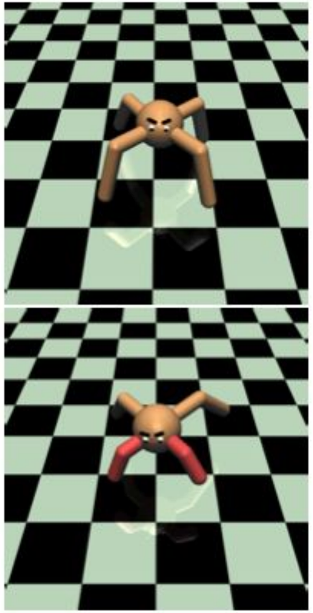
- Model-free imitation learning algorithm
- Efficient in terms of expert data
- Uses Trust Region Policy Optimization (TRPO)
- Not sample efficient in terms of environment interactions

Adversarial Inverse Reinforcement Learning (AIRL)

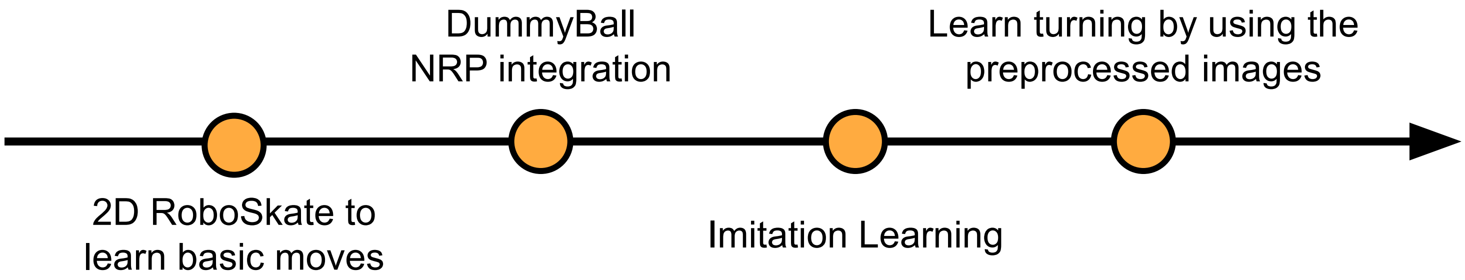
- Simultaneous learning of the reward function and value function
  - Make use of the efficient adversarial formulation
  - Recover a generalizable reward function

- Robust to changes in the environment dynamics

- Compared to GAIL
  - Similar performance in traditional imitation learning setups
  - Outperforms in transfer learning setups



Future work


DummyBall
NRP integration
Learn turning by using the
preprocessed images
2D RoboSkate to
learn basic moves
Imitation Learning

Schedule

22.04. — Kick-off presentation

Simple environment ready to start training

Pipeline for reinforcement learning

Results in simple environment (move forward)

01.06. — Midterm presentation

Advanced environment ready

Oriented movement in advanced environment

15.07. — Final presentation

Responsibilities

	Preliminary Tasks
Michelle Bettendorf	DummyBall, preprocessing images
Gintautas Palinauskas	DummyBall, preprocessing images, teleoperation, DockerFile
Meriç Sakarya	RoboSkate Reward functions, server training
Batuhan Yumurtacı	RoboSkate Reward functions, server training
Finn Süberkrüb	Vectorized Environment, Roboskate learning algorithm, Cloud administration and infrastructure