pip install robot
The Future of AI Development for Robotics

Steven Rice
University of Windsor

University of Windsor
School of Computer Science

About Me

- PhD in Computer Science at the University of Windsor
  - MSc in Computer Science
  - BSc in Computer Science
  - BCom in Business Administration
- Simulations and Digital Twins
- Robotics
- Game Development

StevenRice.ca

University of Windsor
School of Computer Science

“Technical advances alone are not impact”

Dr. Marcus Brubaker – Google DeepMind

University of Windsor
School of Computer Science

Stan: Software for Bayesian Data Analysis

github.com/stan-dev/stan

University of Windsor
School of Computer Science

Stan: Software for Bayesian Data Analysis

- Released in 2011.
- Contained algorithms that were decades old.
    - Why build this library?
        - Nobody else did!
- Improved algorithms existed but rarely used.
    - No friendly tools for developers.
    - Stan changed that!

AI for Robotics

- Reaching this point for robotics.
  - Many amazing AI uses of robotics.
  - Rarely applied outside of research labs.
- When will we have a “Stan for robotics”?
  - What will it need to do?
  - How can it be adopted?

University of Windsor
School of Computer Science

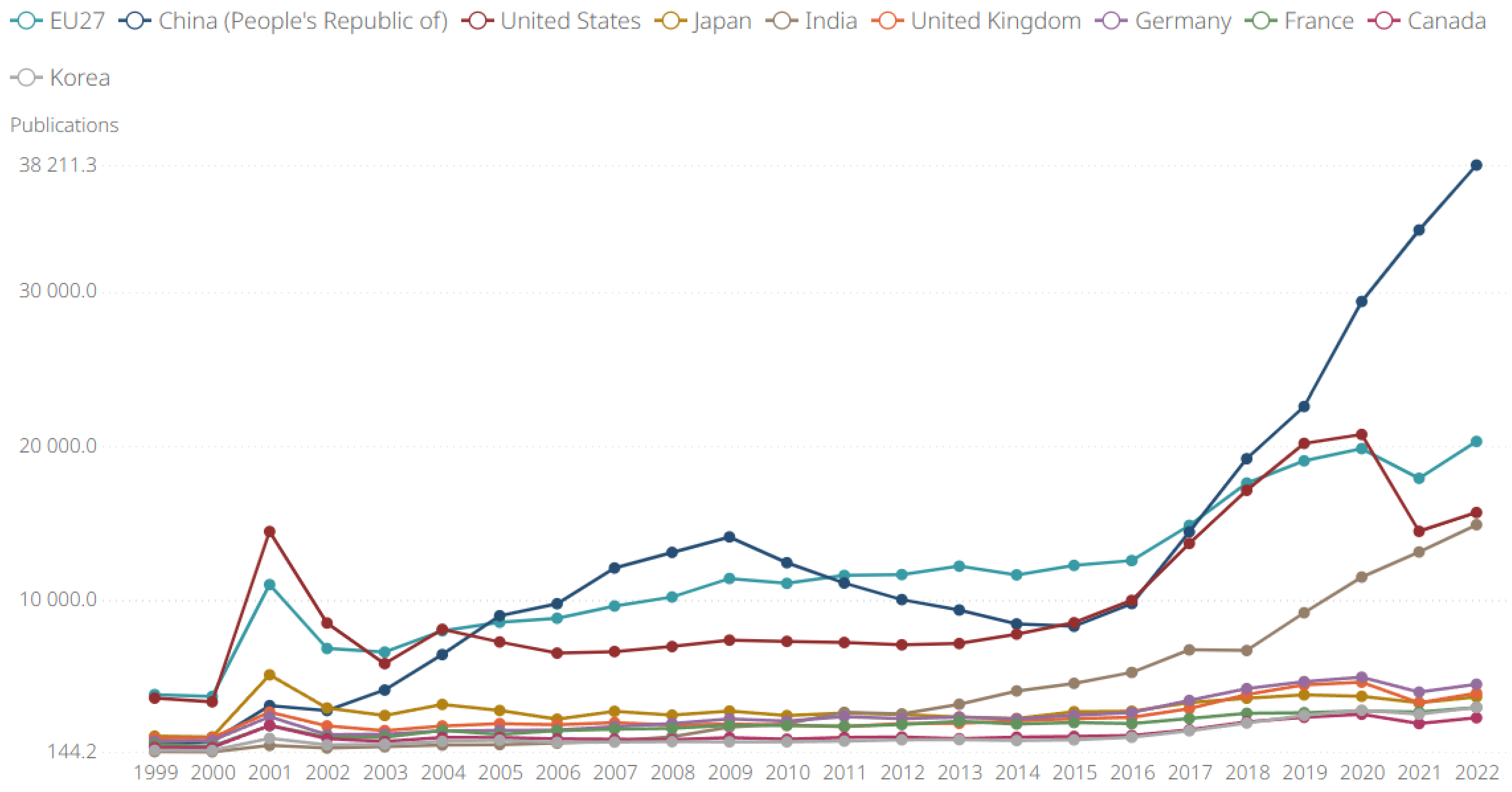
Outline

- AI Development
- Robotics Development
  - AI for Robotics
    - Bottlenecks
- Emerging Tools
- Future

University of Windsor
School of Computer Science

AI Development over Time

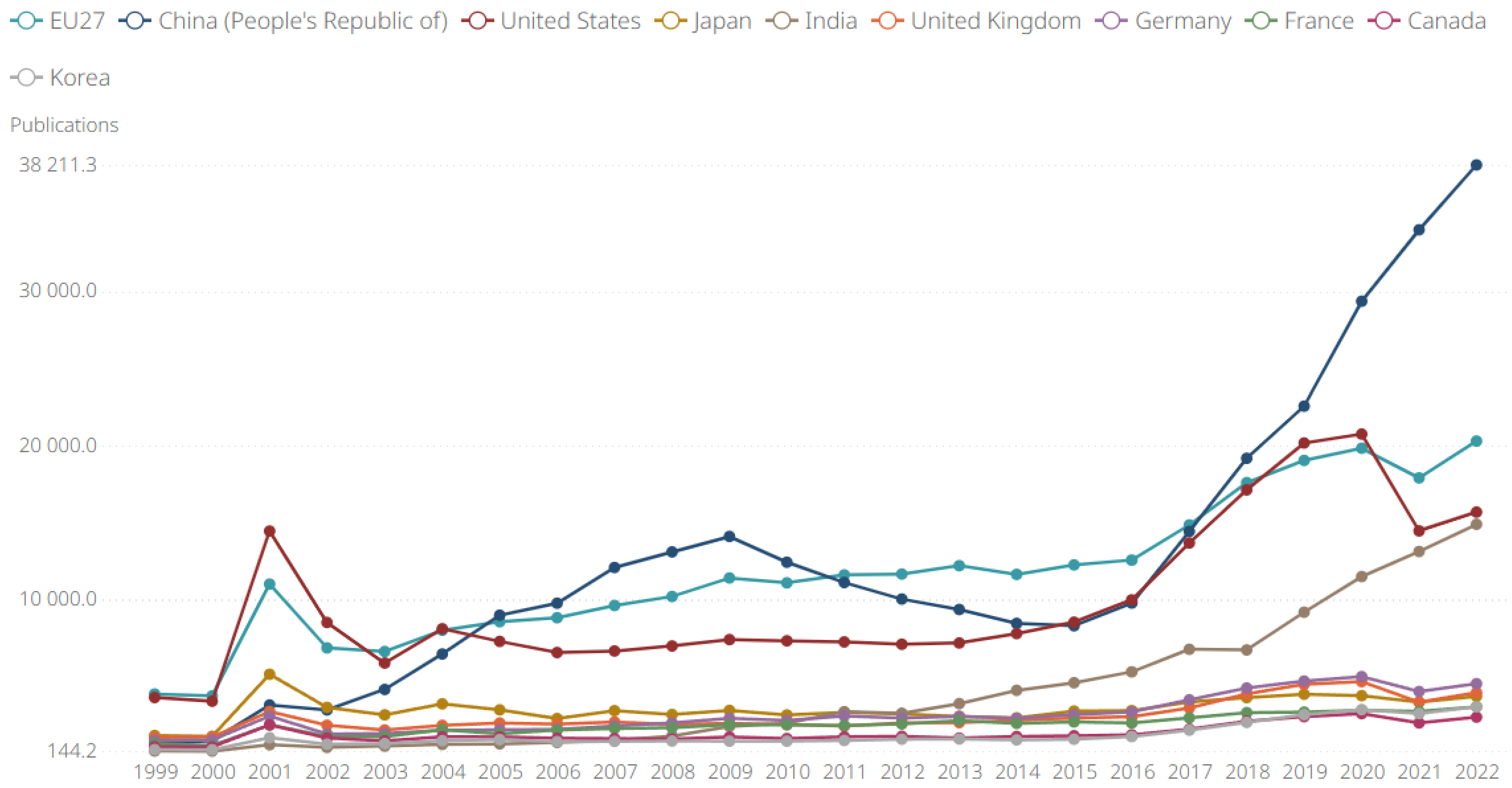
- Continually growing.
  - Across all domains.
- What's the cause of this?
  - Hype?
  - We getting smarter?
  - Hardware?
  - Software?



OECD.AI (2025), data from OpenAlex – oecd.ai

AI Tool Releases

- Scikit-learn – 2010
- TensorFlow – 2015
- PyTorch – 2016



OECD.AI (2025), data from [OpenAlex](#) – [oecd.ai](#)

What Was New?

- Neural Networks – 1940s
- Deep Learning – 1970s
- Backpropagation – 1980s
- Convolutional Neural Networks – 1980s
- Recurrent Neural Networks – 1980s
- These libraries didn't invent any new algorithms!
  - Just made them more accessible.

AI Development – Then Versus Now

- **Then**
  - Implementing specific algorithms.
  - Niche and scattered libraries.
  - Multiple languages.
- **Now**
  - Just call a library.
  - In the language you want.
  - Flip a switch to toggle hardware.

```
import torch

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

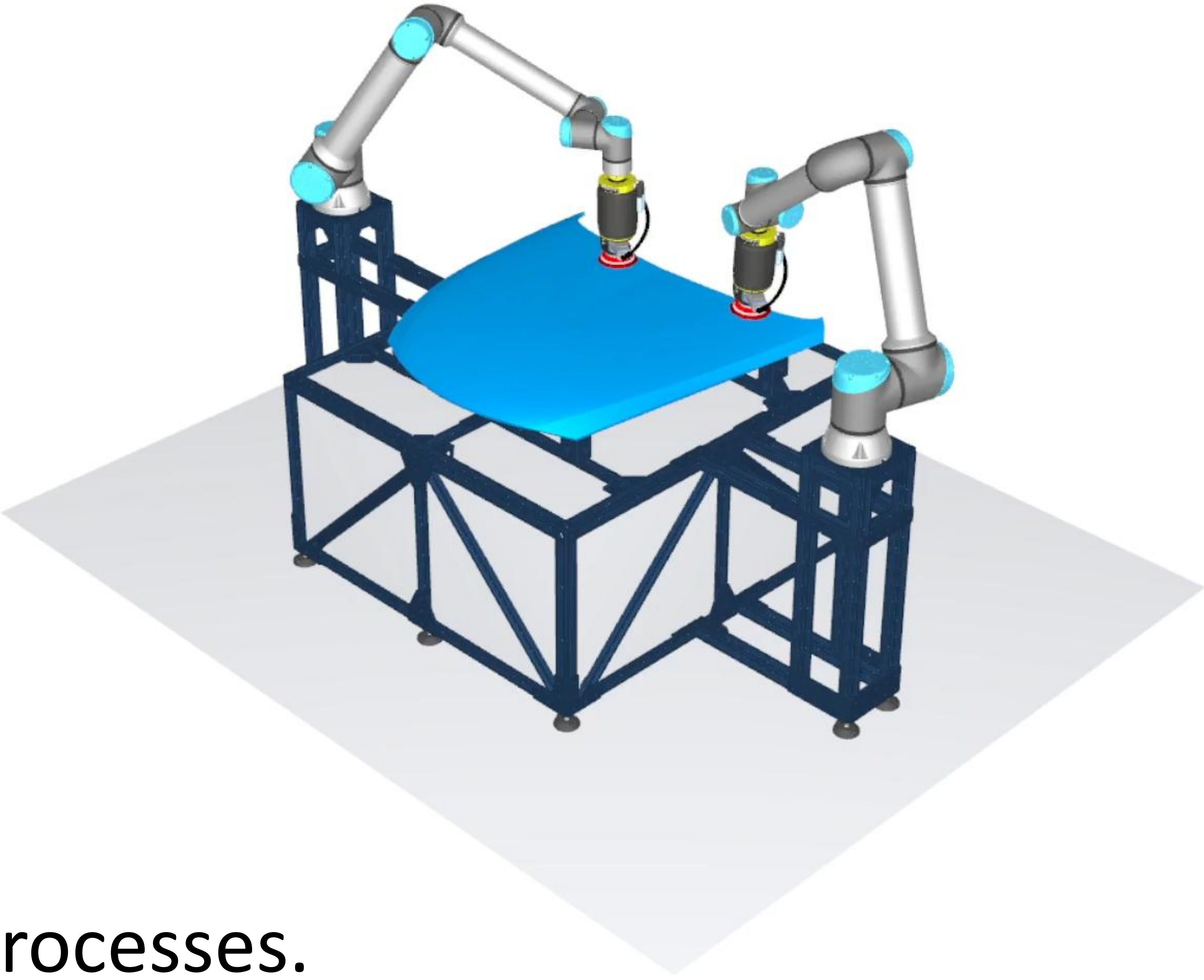
Robotics Development

- General
- AI for Robotics
  - Bottlenecks

University of Windsor
School of Computer Science

Traditional Robotics Development

- Set programs.
  - Sequential move commands.
    - Linear – MoveL
    - Joint – MoveJ
    - Circular – MoveC
- Vendor specific.
- RoboDK.
  - Vendor-agnostic.
- Enough for basic manufacturing processes.
  - Where might this need more?



RoboDK – robodk.com

Advancements in Robotics

- Not just a robot!
  - Sensors.
    - Cameras.
    - LIDAR.
  - Multiple systems.
    - Mobile base.
    - Mounted arm.
  - Dynamic environments.
    - Internals and externals.
- Can't create concrete move programs!

JetRover – Hiwonder – hiwonder.com/products/jetrover

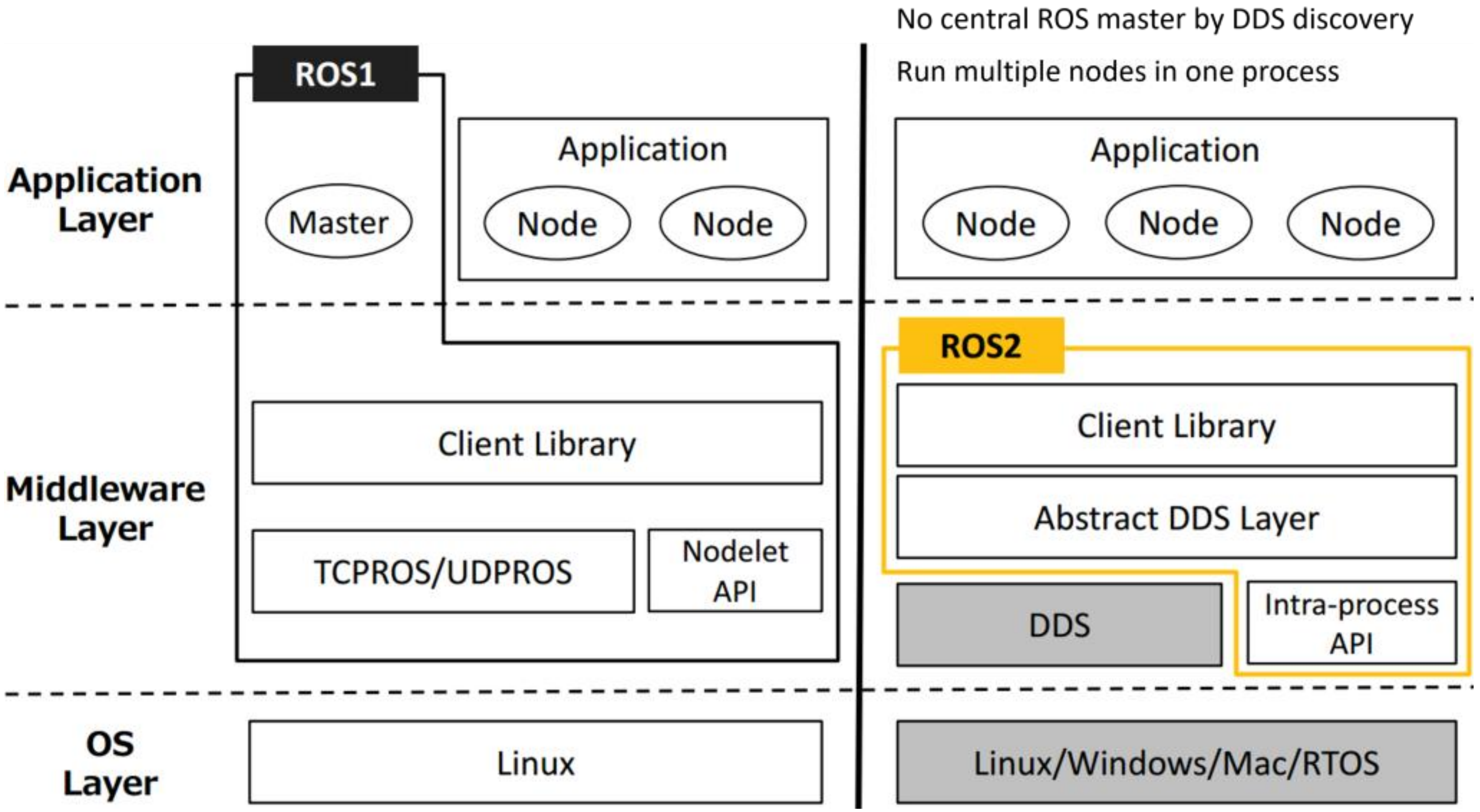
Robot Operating System (ROS)

- Designed for advanced robotics development.
- Facilities communication between components.
    - Robots.
    - Sensors.
- Tailored for C++.
    - Expanded to Python.
- Many community packages.
- ros.org

University of Windsor
School of Computer Science

ROS and ROS2

- ROS has reached end of life.
- ROS2 has better modularity.
  - Multi-robot support.
- ROS packages not compatible with ROS2.
  - Split support.



Dr. Hang Cui - hangpersonal.com/2024/12/07/comparison-of-ros1-and-ros2

AI for Robotics

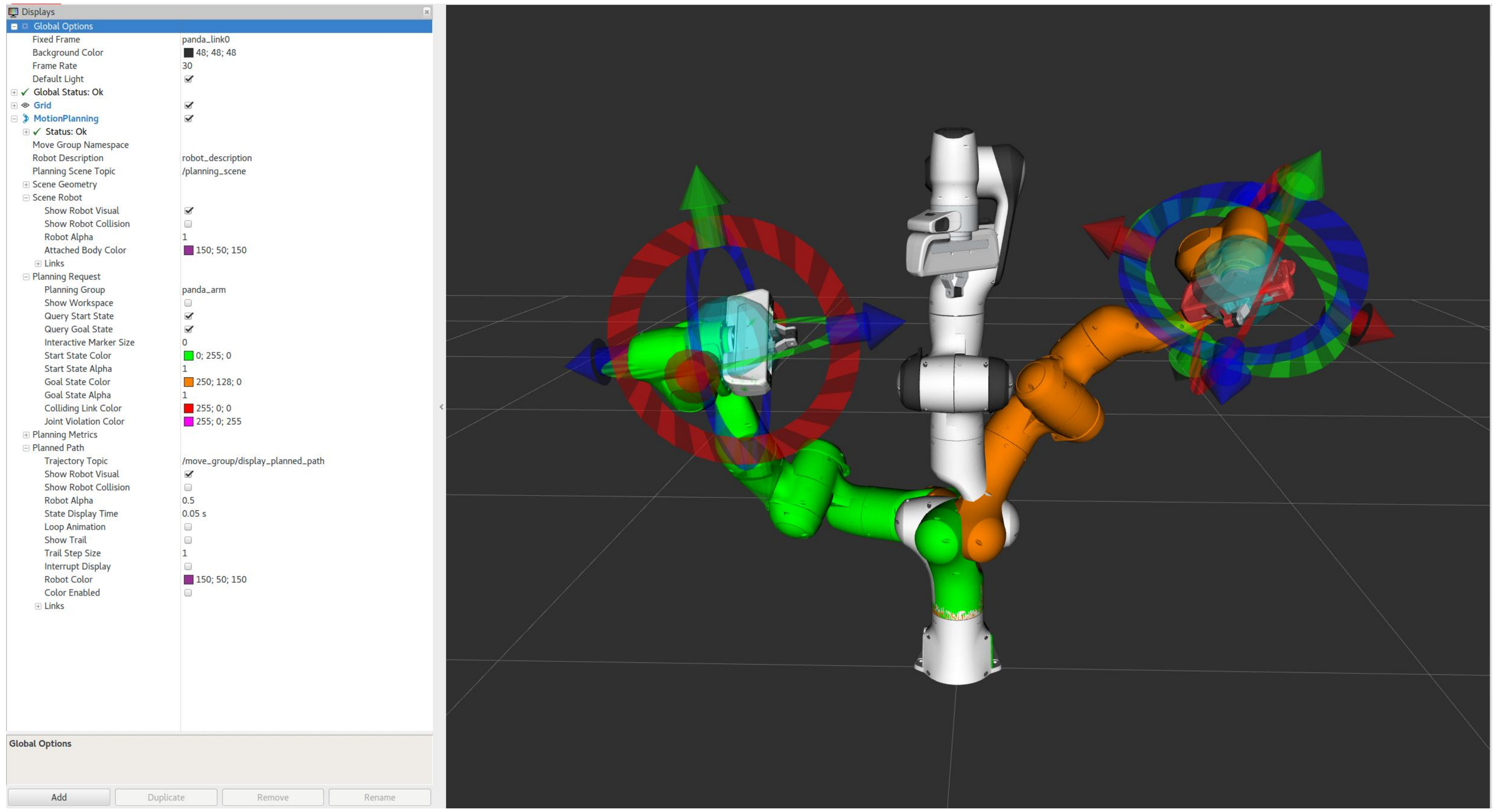
- What do AI models in general need?
  - Data.
  - Training time.
  - Training environment.
- Robotics are expensive.
  - And can be dangerous!
- How can we test robotics?
  - Trial-and-error in the real world?
  - Train reinforcement learning agents in the real world?

University of Windsor
School of Computer Science

Simulation for Robotics AI

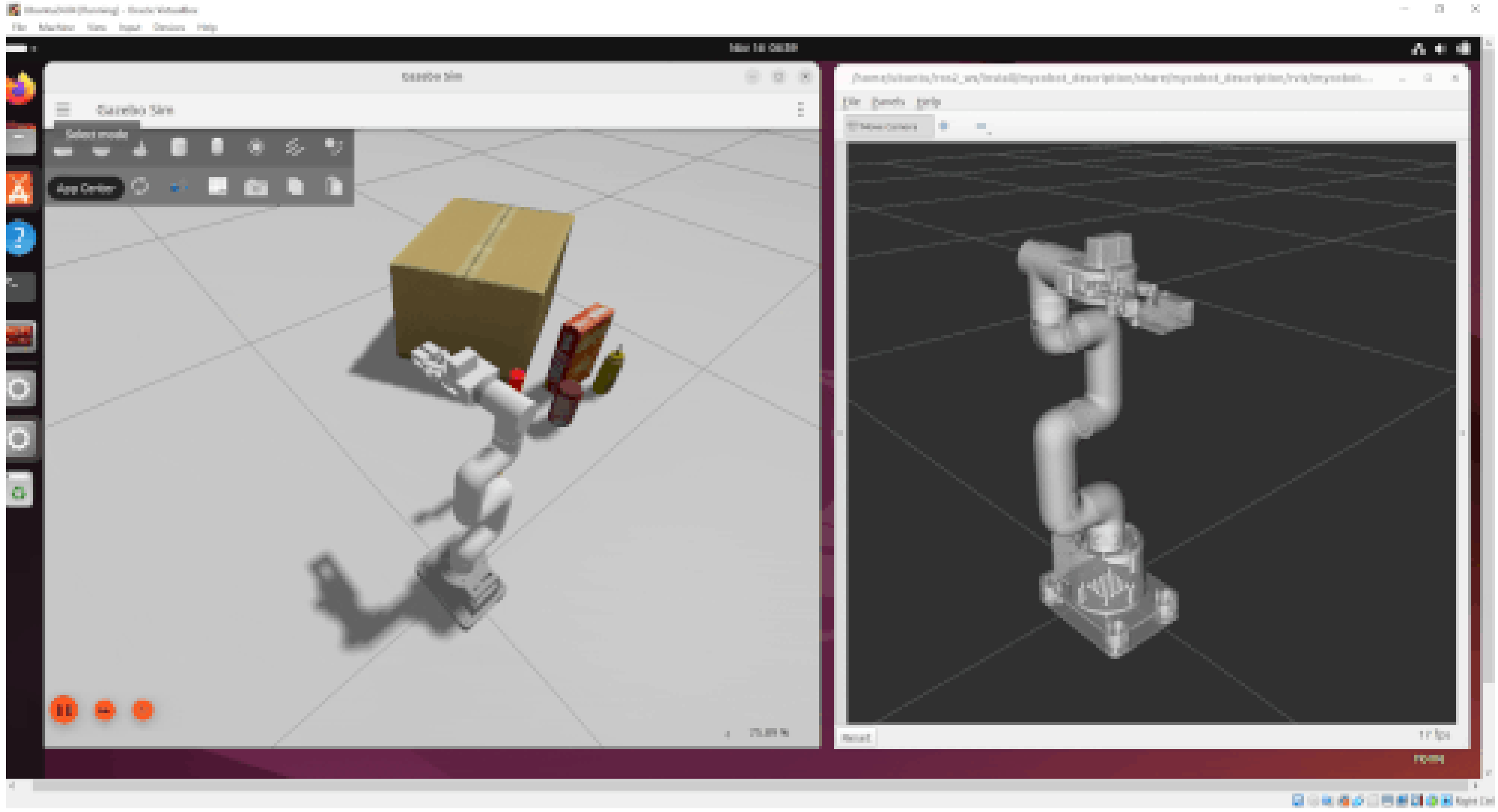
- Train and test in simulation!
- Requirements?
  - Accurate physics.
    - Internal and external.
    - Sensor modelling.
  - High-fidelity visuals.
    - Camera intrinsics.

ROS Visualization



rviz

wiki.ros.org/rviz



Gazebo

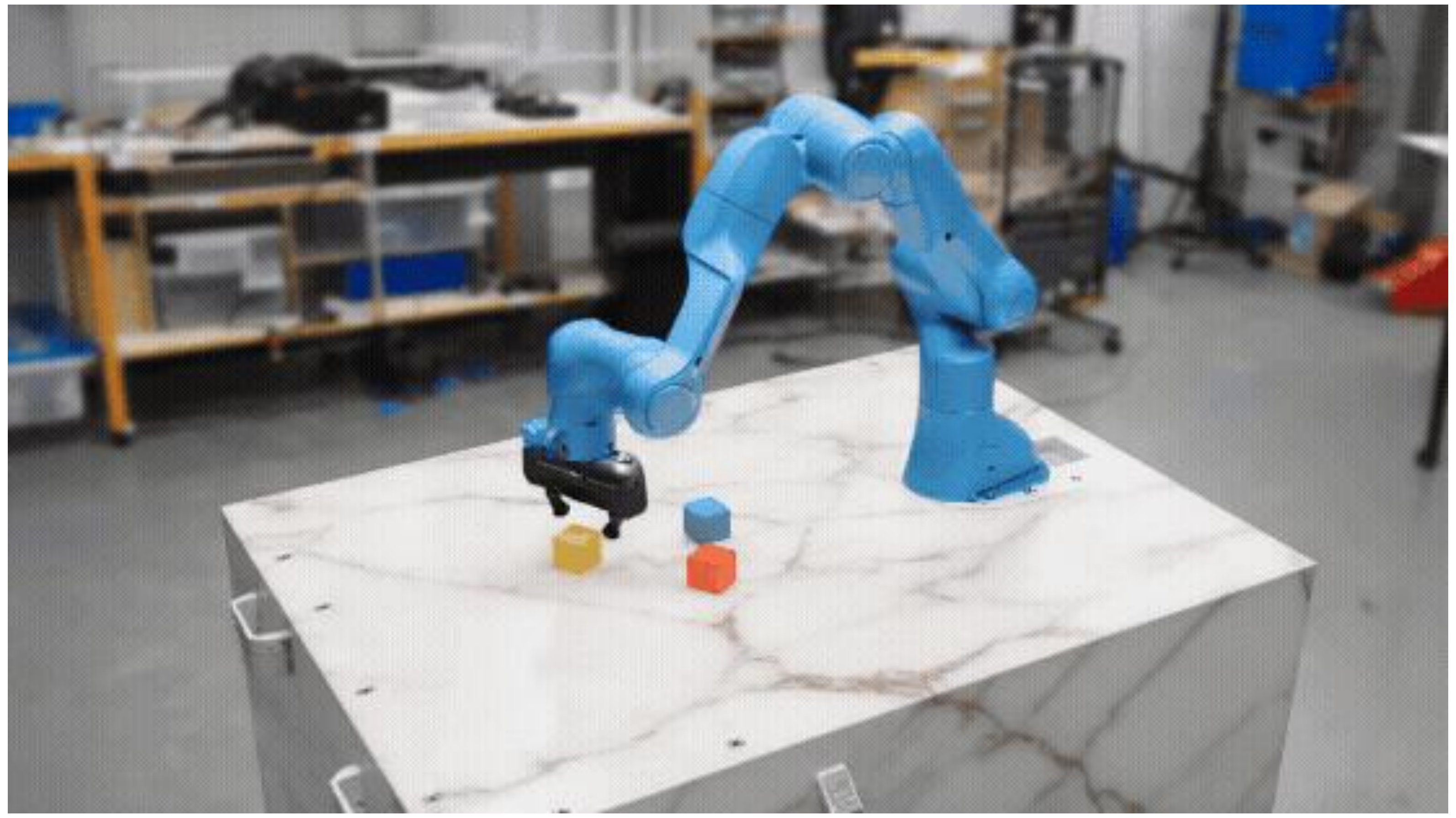
gazebosim.org

University of Windsor
School of Computer Science

ROS Visualization

- What if you need higher fidelity or physical accuracy?
  - Communicate with another application.
- Any issues with these?
  - Extra communication.
  - Duplicated logic.
  - Additional languages.
  - Increased complexity.
  - Multiple scene setups.

Unity – unity.com
ROS# – github.com/siemens/ros-sharp



NVIDIA Isaac Sim – developer.nvidia.com/isaac/sim

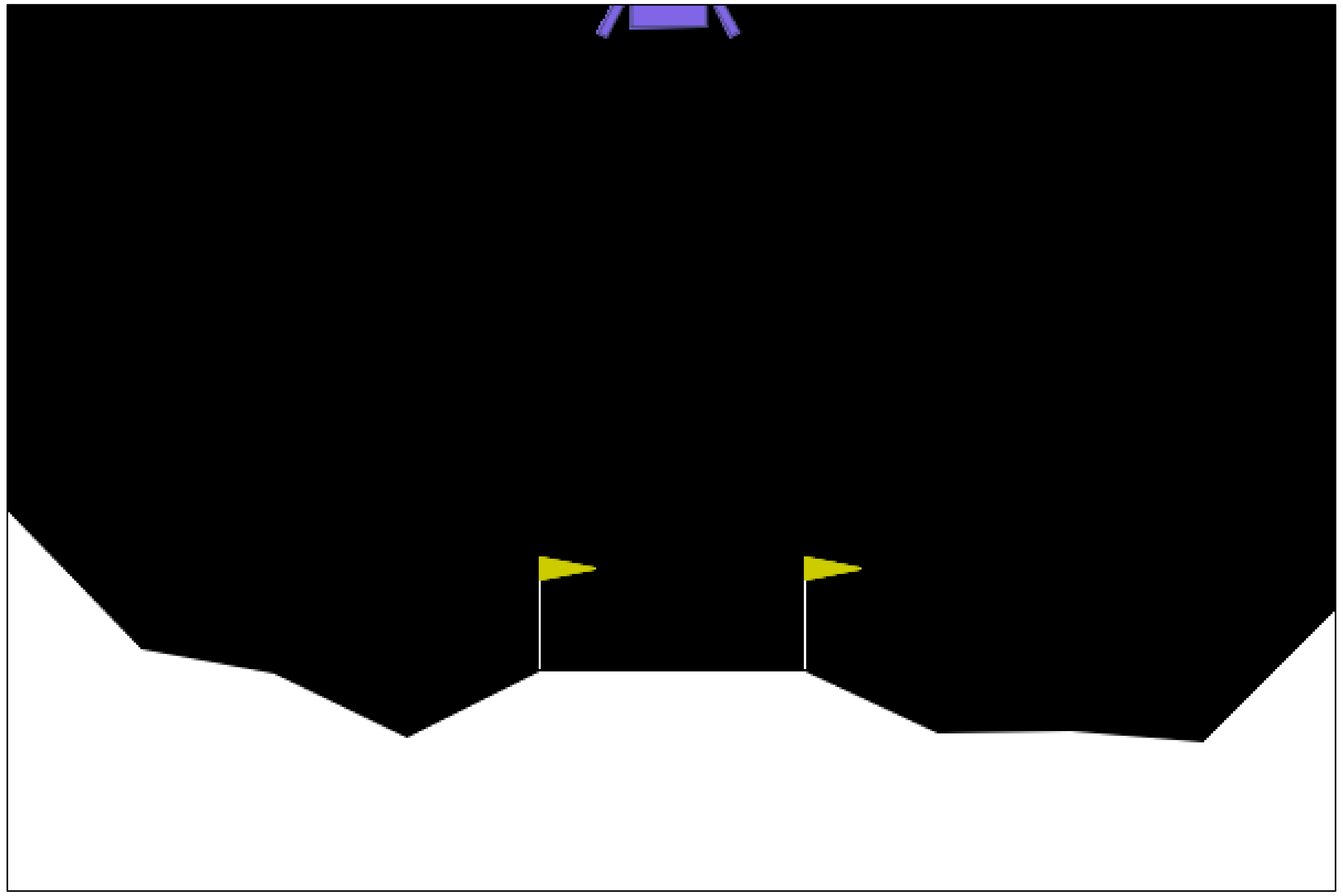
AI Development with ROS

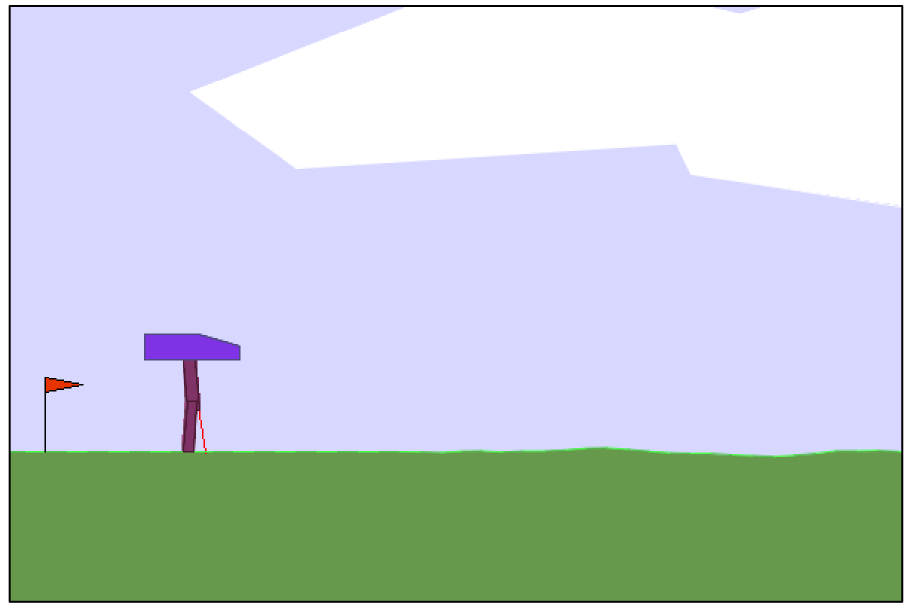
- These don’t impact me!
  - My use case doesn’t need high fidelity.
  - I have the resources for external integration.
- What else could hurdle AI development with ROS?
  - AI training.

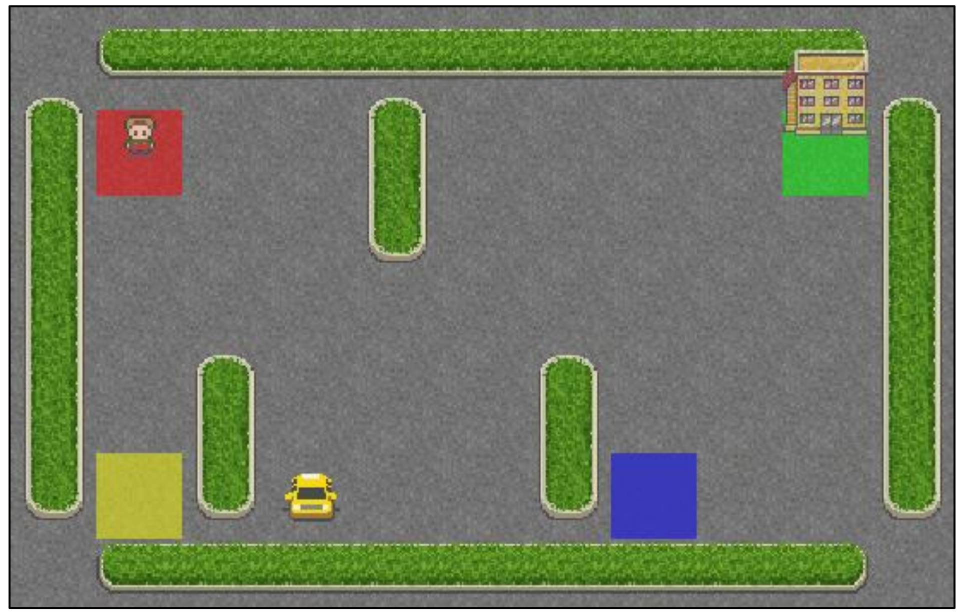
University of Windsor
School of Computer Science

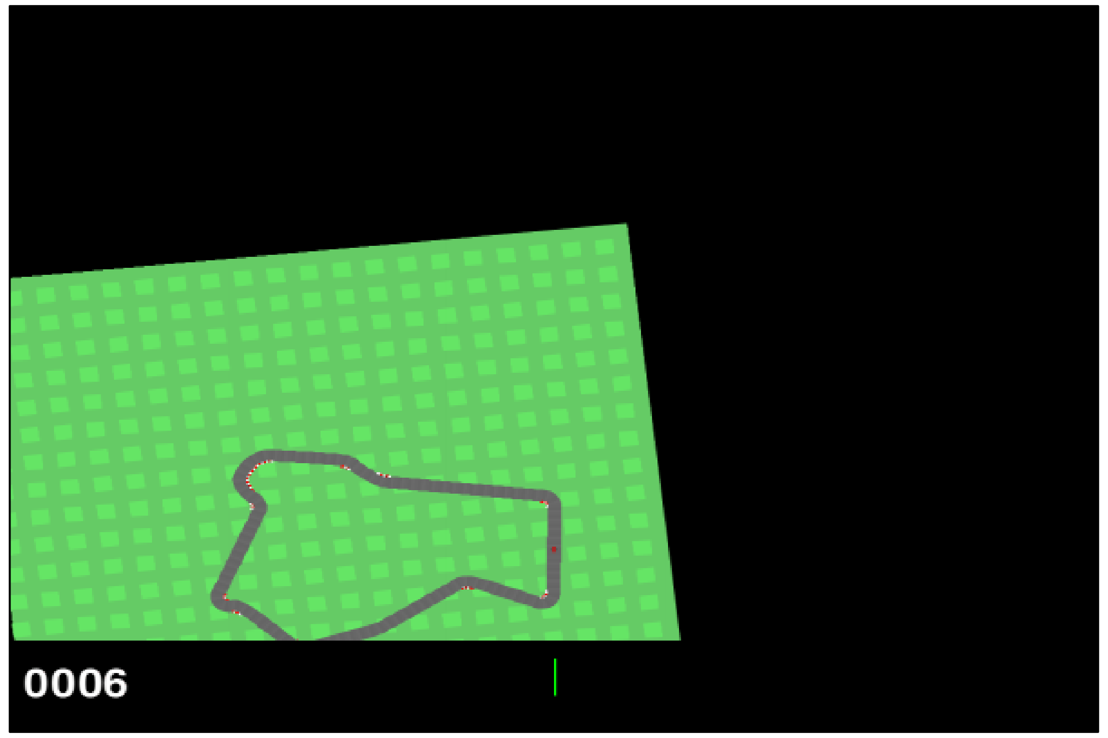
Gymnasium

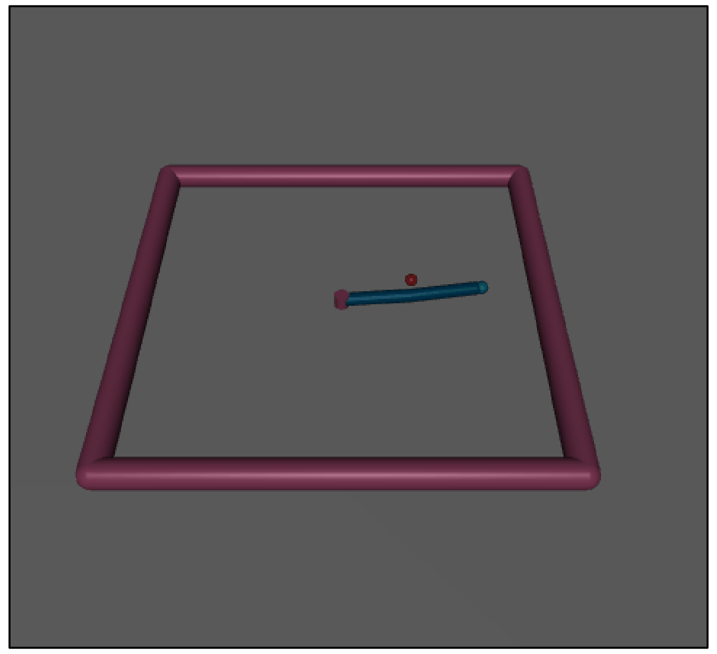
- gymnasium.farama.org
- Originally OpenAI Gym.
- Significance?
  - Easy training loops!
    - Reset and step environment.









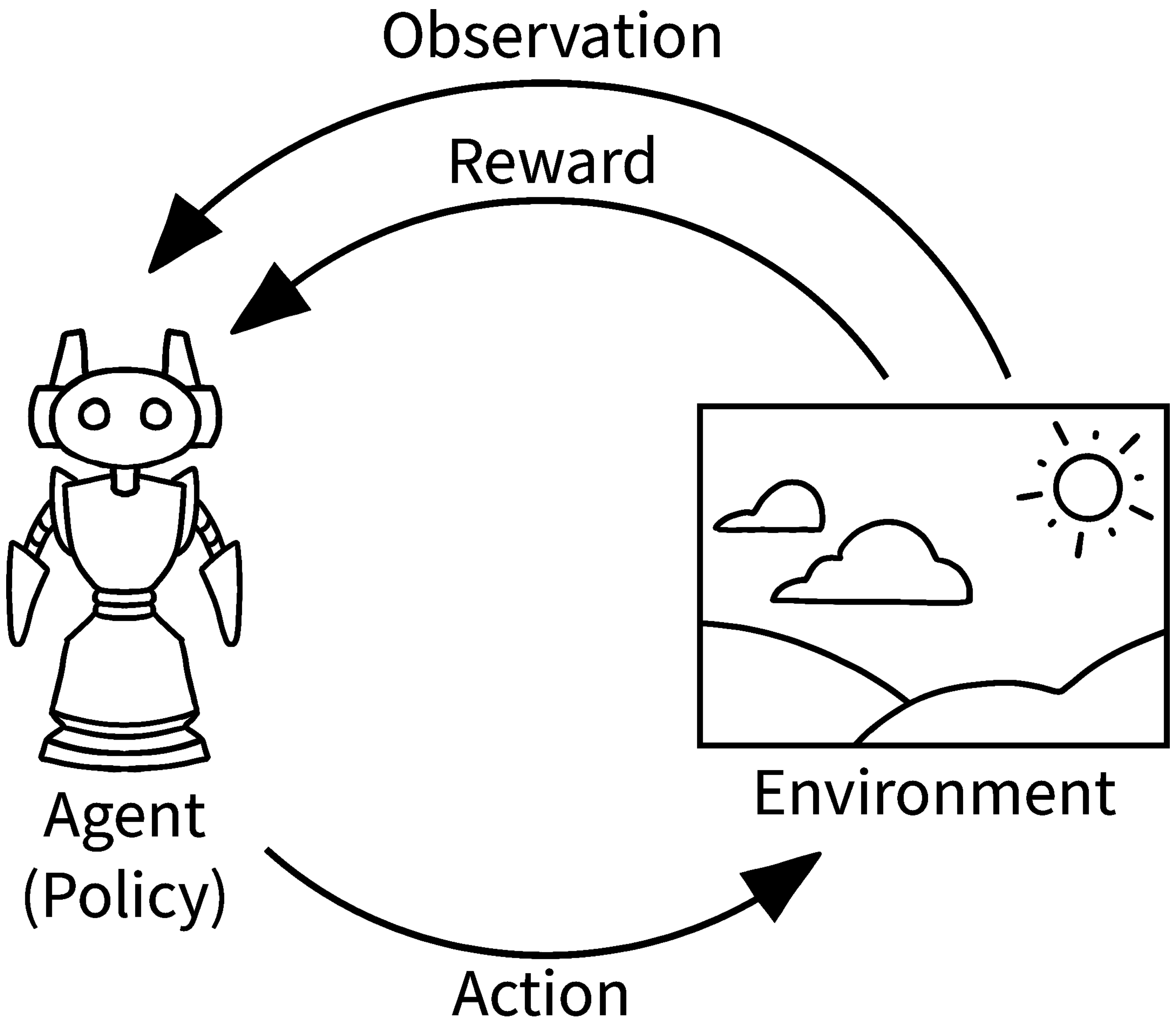


Gymnasium

```python
def reset(self, seed: Optional[int] = None, options: Optional[dict] = None):
    """Start a new episode.

    Args:
        seed: Random seed for reproducible episodes
        options: Additional configuration (unused in this example)

    Returns:
        tuple: (observation, info) for the initial state
    """
    # IMPORTANT: Must call this first to seed the random number generator
    super().reset(seed=seed)

    # Randomly place the agent anywhere on the grid
    self._agent_location = self.np_random.integers(0, self.size, size=2, dtype=int)

    # Randomly place target, ensuring it's different from agent position
    self._target_location = self._agent_location
    while np.array_equal(self._target_location, self._agent_location):
        self._target_location = self.np_random.integers(
            0, self.size, size=2, dtype=int
        )

    observation = self._get_obs()
    info = self._get_info()

    return observation, info
```

```python
def step(self, action):
    """Execute one timestep within the environment.

    Args:
        action: The action to take (0-3 for directions)

    Returns:
        tuple: (observation, reward, terminated, truncated, info)
    """
    # Map the discrete action (0-3) to a movement direction
    direction = self._action_to_direction[action]

    # Update agent position, ensuring it stays within grid bounds
    # np.clip prevents the agent from walking off the edge
    self._agent_location = np.clip(
        self._agent_location + direction, 0, self.size - 1
    )

    # Check if agent reached the target
    terminated = np.array_equal(self._agent_location, self._target_location)

    # We don't use truncation in this simple environment
    # (could add a step limit here if desired)
    truncated = False

    # Simple reward structure: +1 for reaching target, 0 otherwise
    # Alternative: could give small negative rewards for each step to encourage efficiency
    reward = 1 if terminated else 0

    observation = self._get_obs()
    info = self._get_info()

    return observation, reward, terminated, truncated, info
```

Gymnasium




```python
# Run `pip install "gymnasium[classic-control]"` for this example.
import gymnasium as gym

# Create our training environment - a cart with a pole that needs balancing
env = gym.make("CartPole-v1", render_mode="human")

# Reset environment to start a new episode
observation, info = env.reset()
# observation: what the agent can "see" - cart position, velocity, pole angle, etc.
# info: extra debugging information (usually not needed for basic learning)

print(f"Starting observation: {observation}")
# Example output: [ 0.01234567 -0.00987654  0.02345678  0.01456789]
# [cart_position, cart_velocity, pole_angle, pole_angular_velocity]

episode_over = False
total_reward = 0

while not episode_over:
    # Choose an action: 0 = push cart left, 1 = push cart right
    action = env.action_space.sample()  # Random action for now - real agents will be smarter!

    # Take the action and see what happens
    observation, reward, terminated, truncated, info = env.step(action)

    # reward: +1 for each step the pole stays upright
    # terminated: True if pole falls too far (agent failed)
    # truncated: True if we hit the time limit (500 steps)

    total_reward += reward
    episode_over = terminated or truncated

print(f"Episode finished! Total reward: {total_reward}")
env.close()
```

AI Training with ROS

- How to implement this training setup?
  - ROS Gazebo Gym – github.com/rickstaa/ros-gazebo-gym
    - Only ROS supported, not ROS2.
    - No maintained ROS2 gym repositories.
  - External communication.
    - NVIDIA Isaac Lab – developer.nvidia.com/isaac/lab
    - Unity ML Agents – github.com/Unity-Technologies/ml-agents
    - External communication hurdles.
    - Extra work to reset ROS components?
  - Custom implementation.
    - Time?
    - Modularity?

NVIDIA Isaac Lab – developer.nvidia.com/isaac/lab

University of Windsor
School of Computer Science

AI for Robotics – Bottlenecks

- Scattered options.
  - ROS versus ROS2.
  - Custom communication.
- Simulation environments.
  - Variable fidelity and physics.
  - Built-in versus external communication?
- Not Pythonic.
  - Tailored for C++ development.
  - No standardized AI training methods.

AI for Robotics – Crossroads

What Do We Need?

- AI-friendly development.
  - Gym-like framework.
  - Python-centric development.
- Need training environments.
  - High fidelity visuals.
  - Accurate physics.
- Seamless Sim2Real.
  - Code once; run on both.

University of Windsor
School of Computer Science

Emerging Tools

- PyRobot – github.com/facebookresearch/pyrobot
  - No ROS2 and required Ubuntu 16.04.
  - No longer under development.
- LeRobot – github.com/huggingface/lerobot
  - Built around agent training.
  - Only Python – Sometimes want C++ for performance.
  - Less modular.
  - Cannot communicate with ROS.
    - Barrier for migration.

Emerging Tools – Ark

- github.com/Robotics-Ark
- Python-first.
  - C++ available if needed.
  - Gym-like framework.
- ROS and ROS2 compatibility.
- Modular.
- Sim2Real.
  - Multiple simulation engines.
  - Toggle to real-world in one line.

University of Windsor
School of Computer Science

Ark – Modularity

Policy
act = policy(obs)

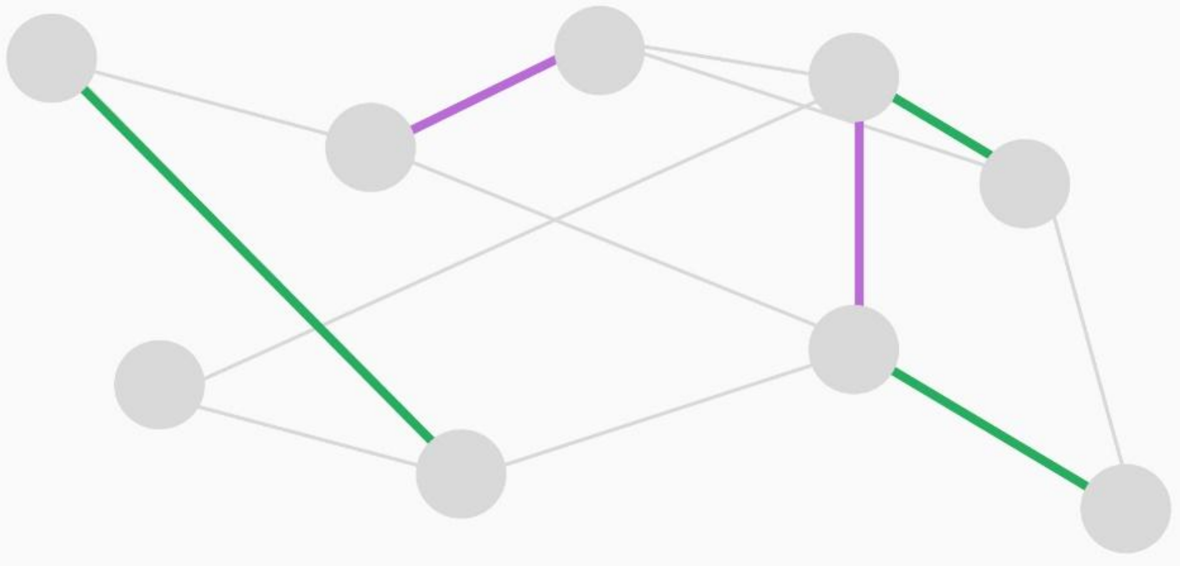
Environment
obs = env.step(act)

Global Config

Action Channels:
- action channel 1
- action channel 2
- ...

Observation Channels:
- obs channel 1
- obs channel 2
- ...

ARK NODE NETWORK



KEY

Node

Action Channels
Observation Channel
Channel

```yaml
# Lists all the hardware components in the experiment
# contains their configurable parameters



# Sets up the simulator with desired specifications
Simulator:
  backend: Pybullet
  ...



# Loads in all the mentioned robots in
# the given positions
Robots:
  - name: Viper
    - position: ...
    - orientation: ...
    - frequency: ...
    ...



# Loads in the sensors in the given positions and
# settings
Sensors:
  - name: Intel Real Sense
    - position: ...
    - orientation: ...
    - frequency: ...
    ...

  - name: Lidar
    - position: ...
    - orientation: ...
    - freqneucy: ...
```


University of Windsor
School of Computer Science

Ark – Pythonic

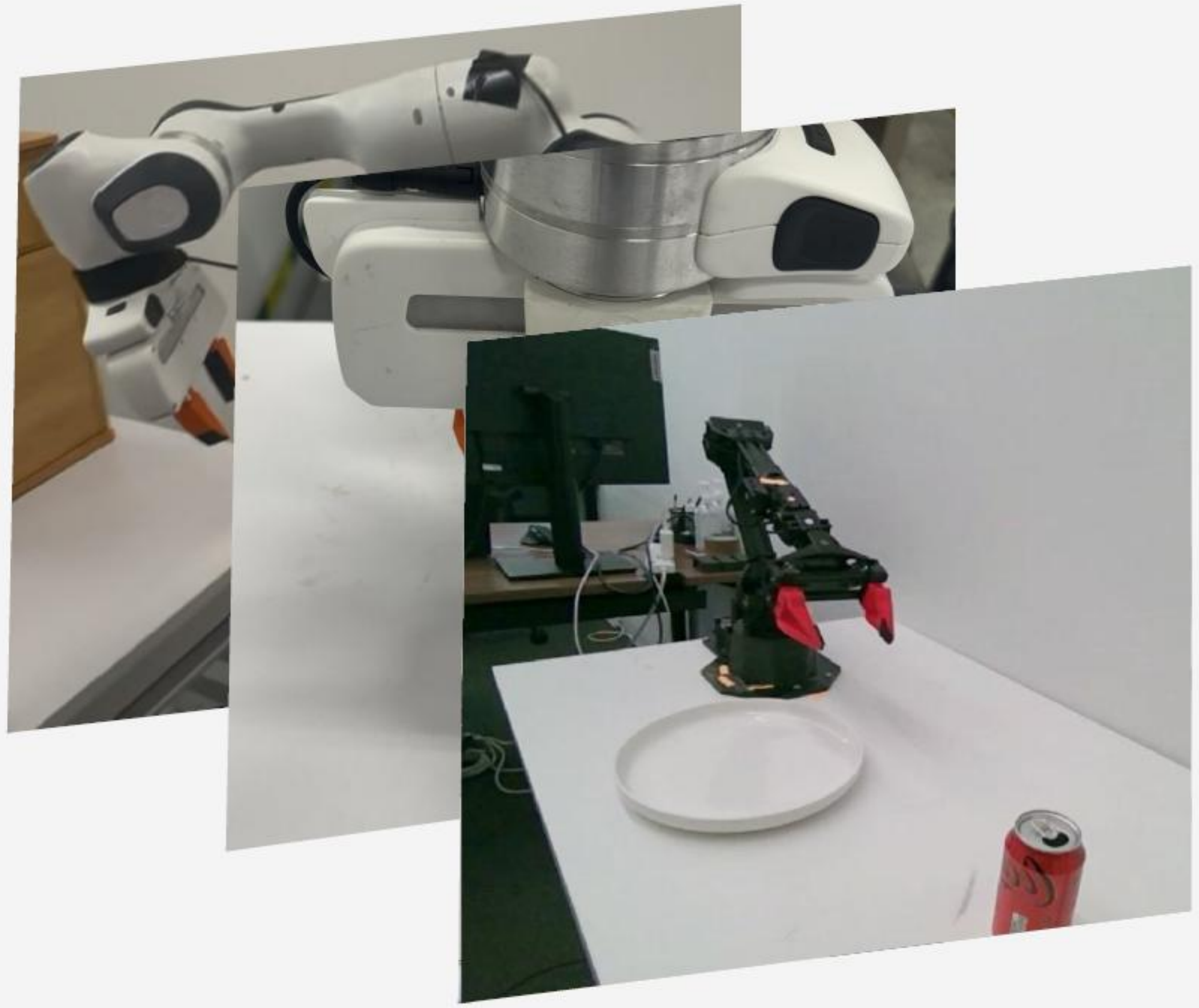
```python
class ViperCokeCan(ArkEnv):
    def __init__(self):
        super().__init__(
            "Viper_Coke_Can",
            observation_channels=[("Viper/joint_commands", joint_commands_t)],
            action_channels=[("Viper/joint_states", joint_states_t)],
        )

    def step(self, action: Any, reward_function=None):
        # Log or print information before taking a step
        log.info(f"Taking step with action: {action}")

        # Takes the action and packes it into Ark Messages
        # Publishes the Ark Messages to the selected channels
        return super().step(action, reward_function)

    def reset(self):
        log.info("Resetting the environment")
        # Sends reset commands to all the robots and sensor
        return super().reset()
```

```python
env = ViperCokeCan()
state = env.reset()
SIM = True  # False

def policy(state):
    ...
    return action

while not done:
    action = policy(state)
    state, reward, done, _ = env.step(action)
```

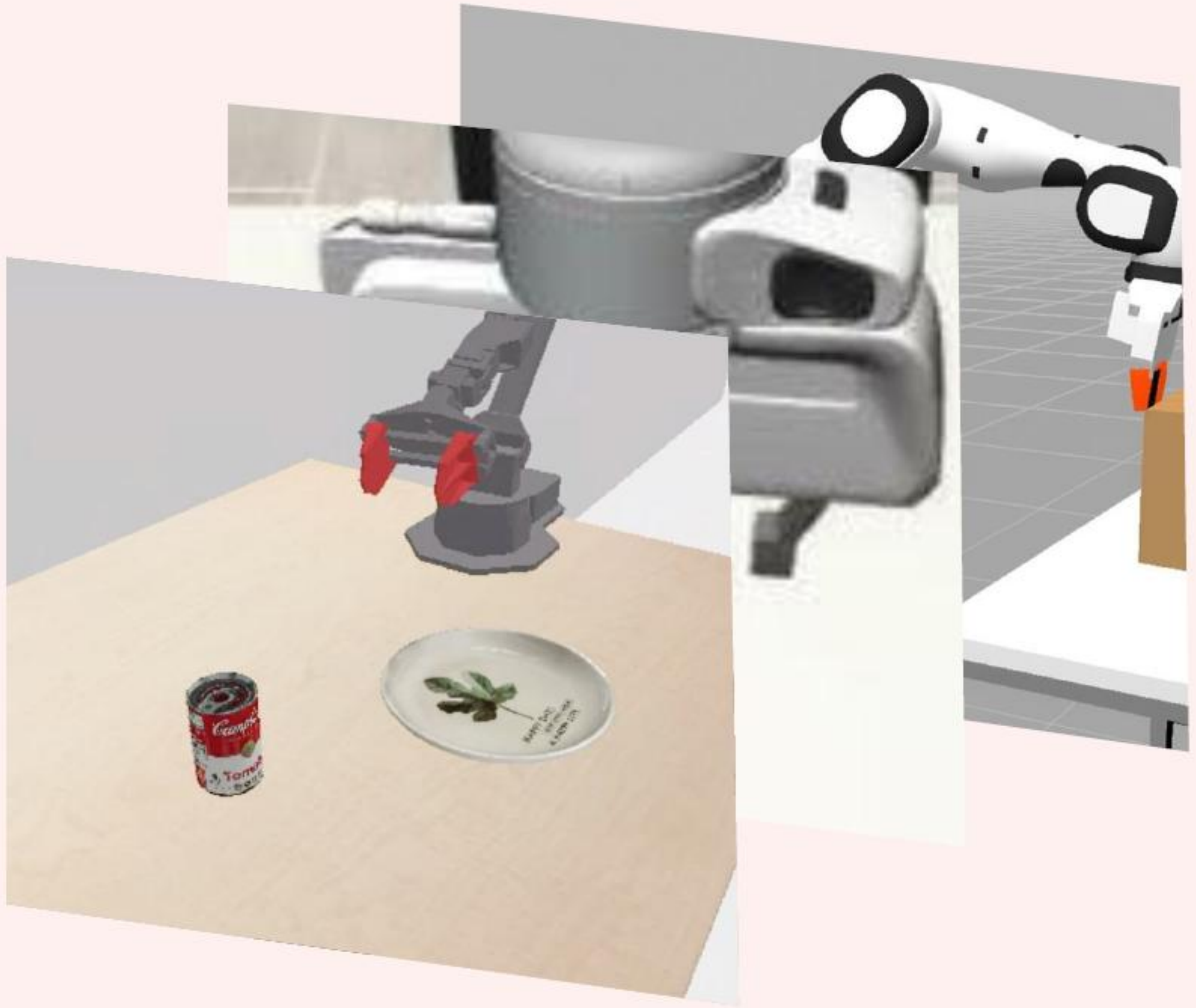

University of Windsor
School of Computer Science

Ark – Sim2Real

Sim = False



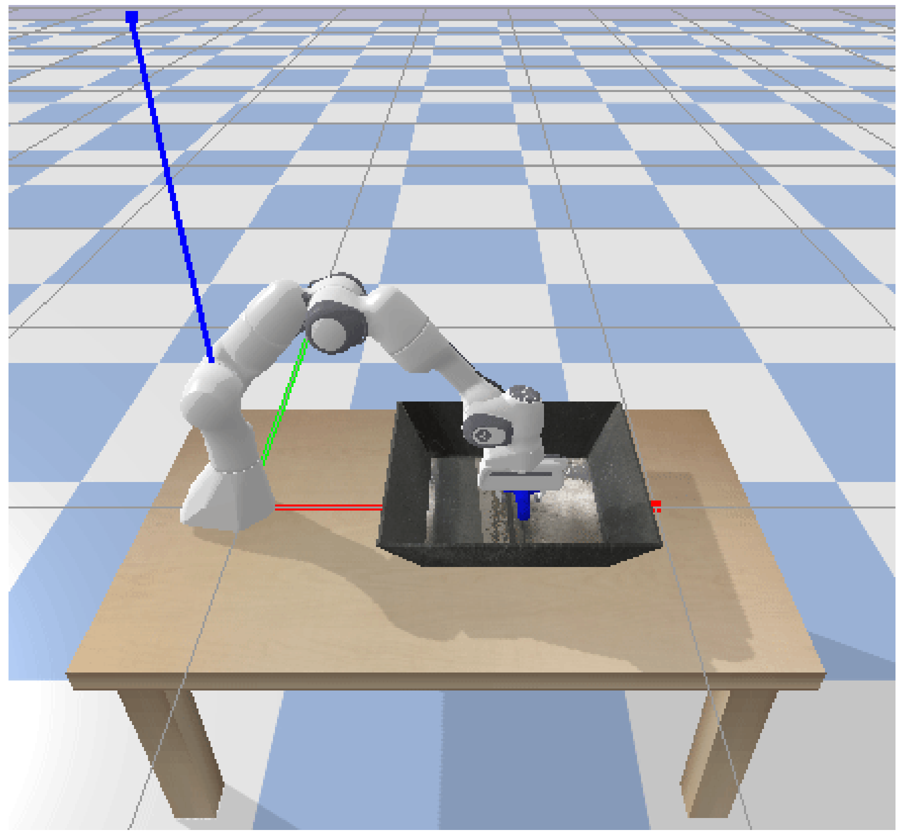
Sim = True



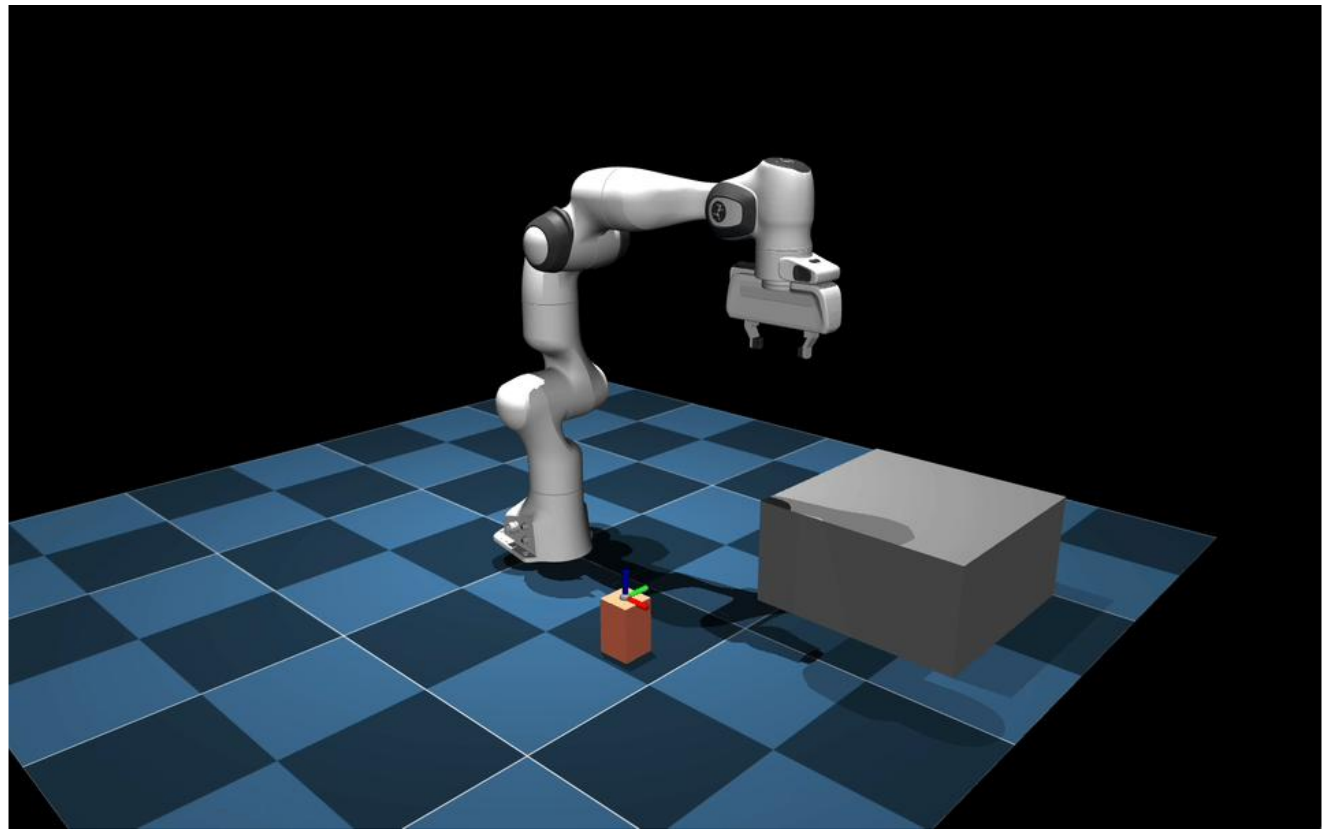
University of Windsor
School of Computer Science

Ark – Sim2Real

- Change backend with a single line.
  - PyBullet, MuJoCo, or Genesis.
  - NVIDIA Isaac Sim planned.



PyBullet – pybullet.org

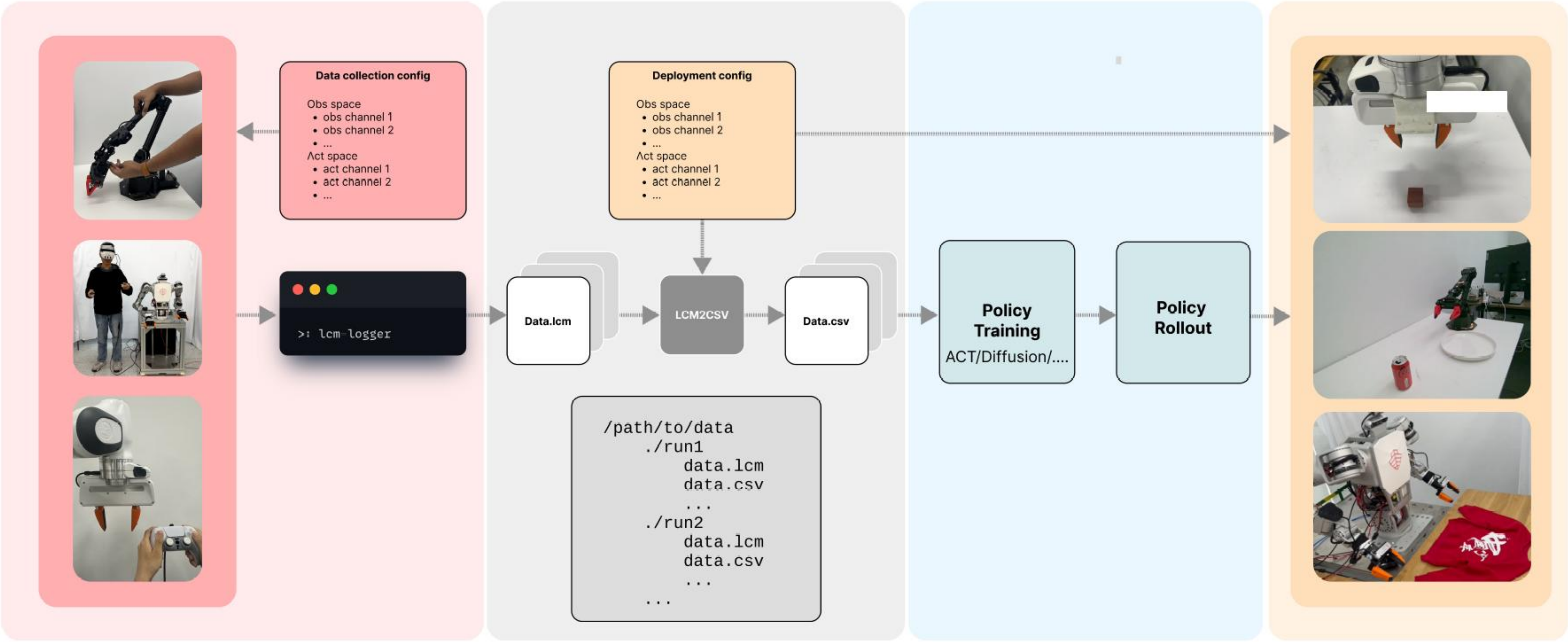


MuJoCo – mujoco.org

Genesis – genesis-embodied-ai.github.io

University of Windsor
School of Computer Science

Ark – Designed for AI



University of Windsor
School of Computer Science

Ark – ROS and ROS2

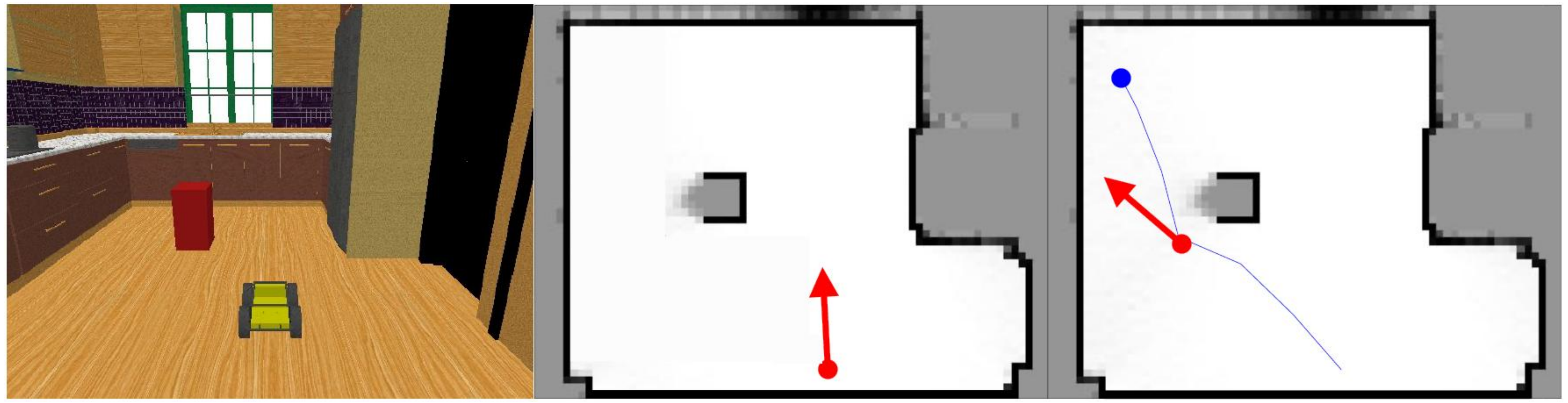
```python
def ros_string_to_ark(ros_msg, ros_channel, ros_type, ark_channel, ark_type):
    """std_msgs/msg/String -> arktypes.string_t"""
    payload = ros_msg.data
    # handle callable vs alias
    msg = ark_type()
    # most ark string types expose `.data`; if yours differs, adjust:
    msg.data = payload
    return msg

def ark_string_to_ros(ark_msg, ros_channel, ros_type, ark_channel, ark_type):
      """arktypes.string_t -> std_msgs/msg/String"""
    out = RosString()
    # if ark uses .data, prefer that; fallback to str(ark_msg)
    out.data = getattr(ark_msg, "data", str(ark_msg))
    return out
```

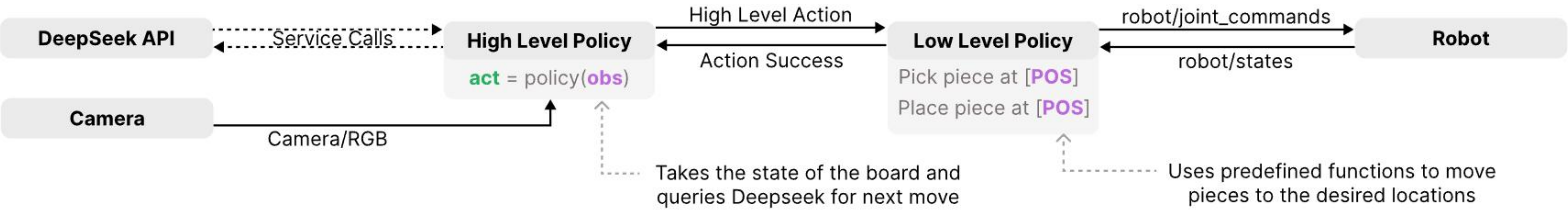
```python
mapping_table = {
    "ros_to_ark": [
        {
            "ros_channel": "/chatter",
            "ros_type": RosString,
            "ark_channel": "ark/chatter",
            "ark_type": string_t,
            "translator_callback": ros_string_to_ark,
        }
    ],
    "ark_to_ros": [
        {
            "ros_channel": "/chatter_mirror",
            "ros_type": RosString,
            "ark_channel": "ark/chatter",
            "ark_type": string_t,
            "translator_callback": ark_string_to_ros,
        }
    ],
}

if __name__ == "__main__":
    # If ArkRos2Bridge handles rclpy init/teardown internally, you can omit these.
    rclpy.init(args=None)
    try:
        bridge = ArkRos2Bridge(mapping_table, node_name="ark_ros2_bridge")
        bridge.spin()          # or rclpy.spin(bridge.node) depending on your class
    finally:
        rclpy.shutdown()
```

Ark – Comparison

	Python	C/C++	Gym	IL	RL	Sim	ROS Dep	ROS Con	PubSub	Sim-Real	Pip	Data tools	Inspect. tools	Limited Embod.
Ark	●	●	●	●		●		●	●	●	●	●	●	
LeRobot	●		●	●	●	●				●	●	●		●
PyRobot	●		●			●	●	●		●				●
ROS 2	●	●				●	NA	NA	●				●	
Orocos		●							●					
YARP		●							●					

Ark – Samples





University of Windsor
School of Computer Science

Ark – Quick Start

```
# Create and activate an environment.
conda create -n ark_env python=3.10
conda activate ark_env

# Install the core framework.
git clone https://github.com/Robotics-Ark/ark_framework.git
cd ark_framework
pip install -e .
cd ..

# Configure types.
git clone https://github.com/Robotics-Ark/ark_types.git
cd ark_types
pip install -e .
cd ..

# Install robots.
git clone https://github.com/Robotics-Ark/ark_franka.git
cd ark_franka
pip install -e .
cd ..
```

Ark – Quick Start

```python
from ark.client.comm_infrastructure.base_node import main
from ark.system.simulation.simulator_node import
SimulatorNode
from ark.tools.log import log

CONFIG_PATH = "config/global_config.yaml"

class MySimulatorNode(SimulatorNode):

    def initialize_scene(self):
        pass

    def step(self):
        pass

if __name__ == "__main__":
    main(MySimulatorNode, CONFIG_PATH)
```

```yaml
simulator:
  name: "franka_simulator"                           # name of the node for ark graph
  backend_type: "pybullet"                           # type of the backend
  node_frequency: 240                                # node frequency in Hz (default 240)
  config:                                            # simulator-specific
    connection_mode: "DIRECT"
    gravity:                                         # default [0, 0, -9.81]
      - 0
      - 0
      - -9.81
    sim_frequency: 240                               # Time step for the simulation (default 240)
  save_render:
    save_path: "render"  # path to save the rendered images
    remove_existing: False                          # remove existing images in the render path
    render_interval: 0.333                          # render every n-th step (default 1)
    overwrite_file: True
    extrinsics:
      look_at: [0.5, 0.0, 0.5]                     # point to look at
      distance: 1.5                               # distance from the point to look at
      azimuth: 0                                 # azimuth angle
      elevation: 0.0                               # elevation angle
    intrinsics:
      fov: 60.0                                    # field of view in degrees
      width: 640                                   # width of the rendered image
      height: 480                                  # height of the rendered image
      field_of_view: 60.0                           # field of view in degrees
      near_plane: 0.01                          # near plane distance
      far_plane: 100.0                             # far plane distance

robots:
  - "robots/franka.yaml"

other:
  - "other/franka_simulator.yaml"                   # simulator configuration
```


University of Windsor
School of Computer Science

Ark – Quick Start

```python
import numpy as np
from arktypes import joint_group_command_t, task_space_command_t, joint_state_t
from arktypes.utils import unpack, pack
from ark.client.comm_infrastructure.instance_node import InstanceNode

SIM = True

class FrankaControllerNode(InstanceNode):

    def __init__(self):
        '''
        Initialize the FrankaJointController.
        This class is responsible for controlling the Franka robot's joints.
        '''
        super().__init__("FrankaJointController")

        if SIM == True:
            self.joint_group_command = self.create_publisher("Franka/joint_group_command/sim", joint_group_command_t)
            self.task_space_command = self.create_publisher("Franka/cartesian_command/sim", task_space_command_t)
            self.state = self.create_listener("Franka/joint_states/sim", joint_state_t)
```

Ark – Quick Start

```python
controller = FrankaControllerNode()

# Set joints.
joint_command = [-0.3, 0.1, 0.3, -1.4, 0.1, 1.8, 0.6, 0]
controller.joint_group_command.publish(pack.joint_group_command(joint_command, "all"))

# Set velocities to open and close the gripper.
gripper_command = [-1] # -1 close, 1 open
controller.joint_group_command.publish(pack.joint_group_command(gripper_command, "gripper"))

# Reach a target.
xyz_command = np.array([0.3, 0.4, 0.8])
quaternion_command = np.array([1, 0.0, 0.0, 0.0]) # xyz-w
gripper = 1.0 # 0.0 close, 1.0 open
controller.task_space_command.publish(pack.task_space_command("all", xyz_command, quaternion_command, gripper))
```


University of Windsor
School of Computer Science

```python
        Initialize the FrankaJointController.
        This class is responsible for controlling the Franka robot's joints.
        '''
        super().__init__("FrankaJointController")

        if SIM == True:
            self.joint_group_command = self.create_publisher("Franka/joint_group_command/sim", joint_group_command_t)
            self.task_space_command = self.create_publisher("Franka/cartesian_command/sim", task_space_command_t)

            self.state = self.create_listener("Franka/joint_states/sim", joint_state_t)

controller = FrankaControllerNode()
```
[1] ✓ 0.4s Python

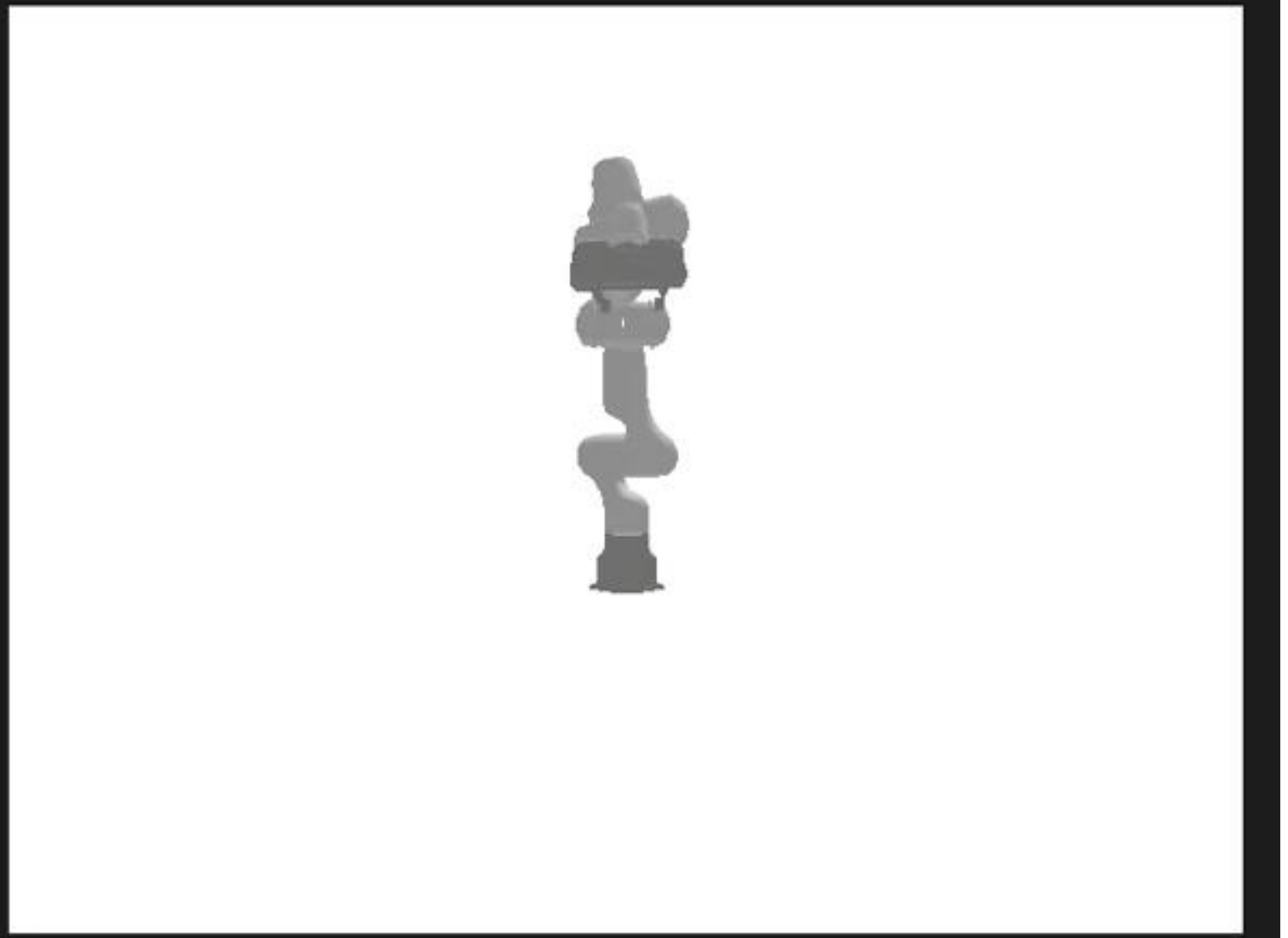
```
[WARNING] [11:40:37.205003] - No global configuration provided. Using default system configuration.
[INFO] [11:40:37.211577] - Service: Successfully registered '__DEFAULT_SERVICE/GetInfo/FrankaJointController_715545ba-ec2d-4673-8b35-74b592
[INFO] [11:40:37.214346] - Service: Successfully registered '__DEFAULT_SERVICE/SuspendNode/FrankaJointController_715545ba-ec2d-4673-8b35-74
[INFO] [11:40:37.216751] - Service: Successfully registered '__DEFAULT_SERVICE/RestartNode/FrankaJointController_715545ba-ec2d-4673-8b35-74
[ERROR] [11:40:37.217436] - Couldn't load config for other 'FrankaJointController'
[OK] [11:40:37.218162] - setup publisher Franka/joint_group_command/sim[joint_group_command_t]
[OK] [11:40:37.219019] - setup publisher Franka/cartesian_command/sim[task_space_command_t]
[OK] [11:40:37.220122] - subscribed to Franka/joint_states/sim[joint_state_t]
None
```

Joint Group Control

```python
joint_command = [-0.3, 0.1, 0.3, -1.4, 0.1, 1.8, 0.6, 0]
# Position Control
controller.joint_group_command.publish(pack.joint_group_command(joint_command, "all"))
```
[4] ✓ 0.0s Python

```python
gripper_command = [1] # -1 close, 1 open
# Velocity Control
controller.joint_group_command.publish(pack.joint_group_command(gripper_command, "gripper"))
```
[5] ✓ 0.0s Python

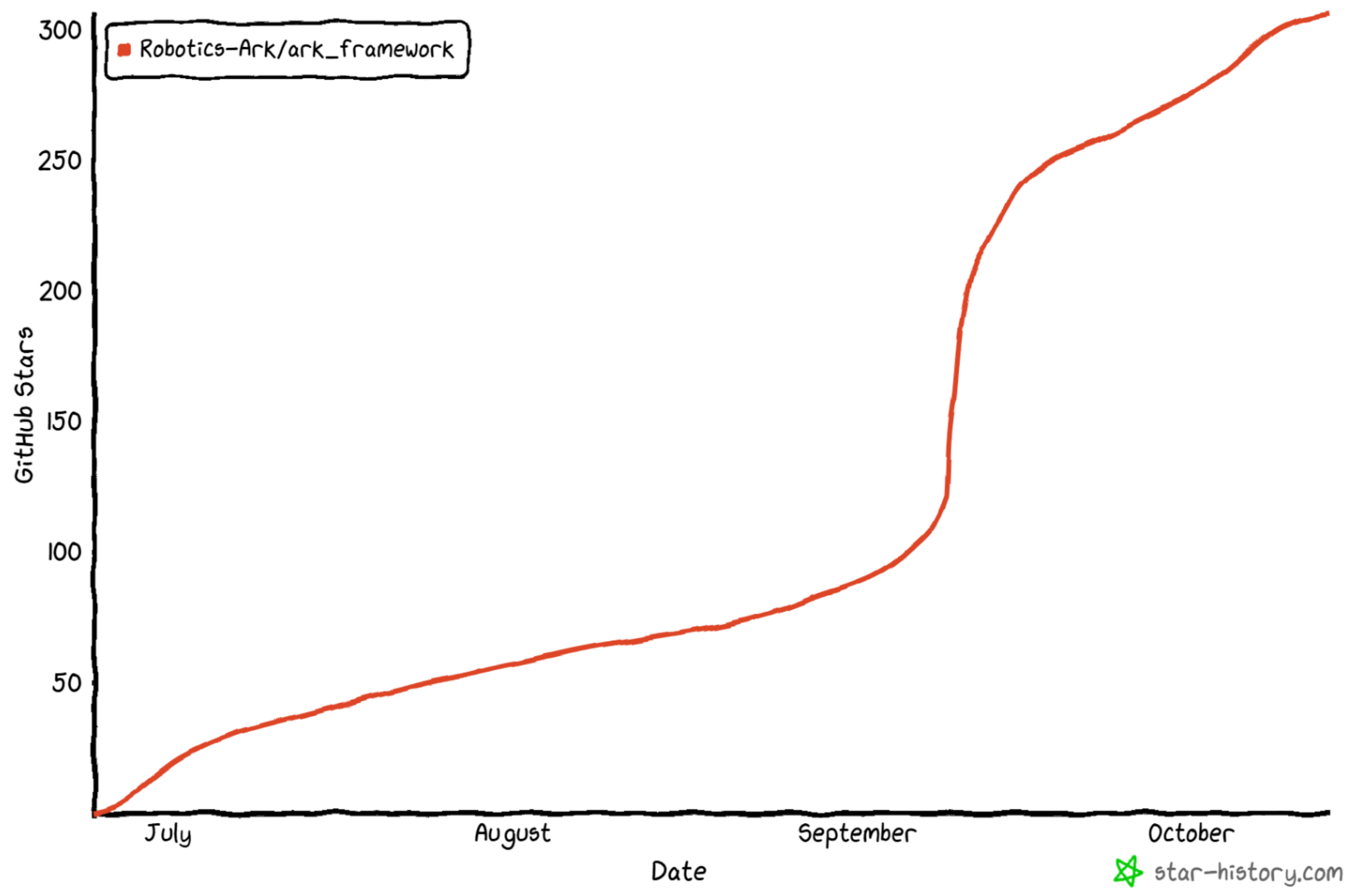
```python
arm_command = [-0.3, 0.5, 0.3, -1.0, 0.1, 1.8, 0.1]
# Position Control
controller.joint_group_command.publish(pack.joint_group_command(arm_command, "arm"))
```
[3] ✓ 0.0s Python

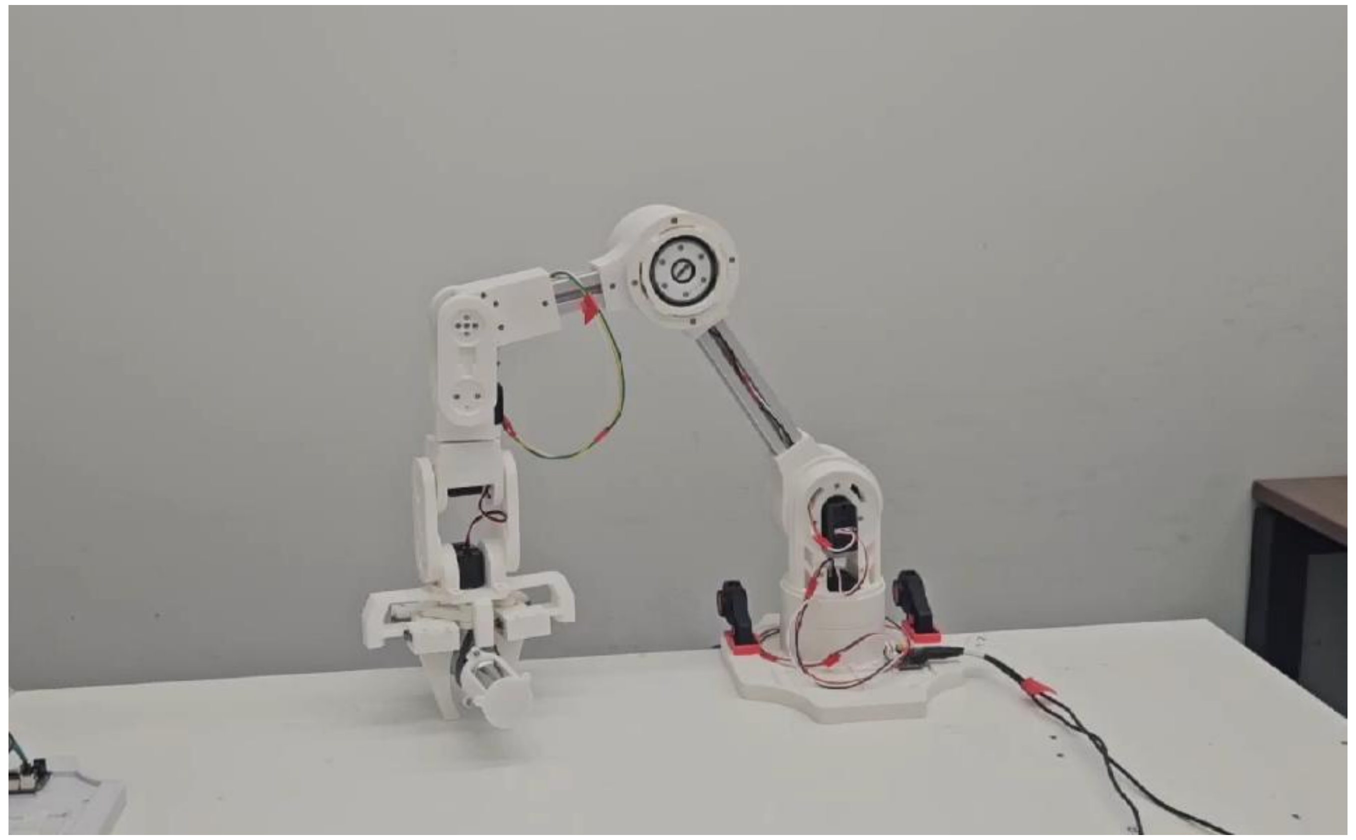


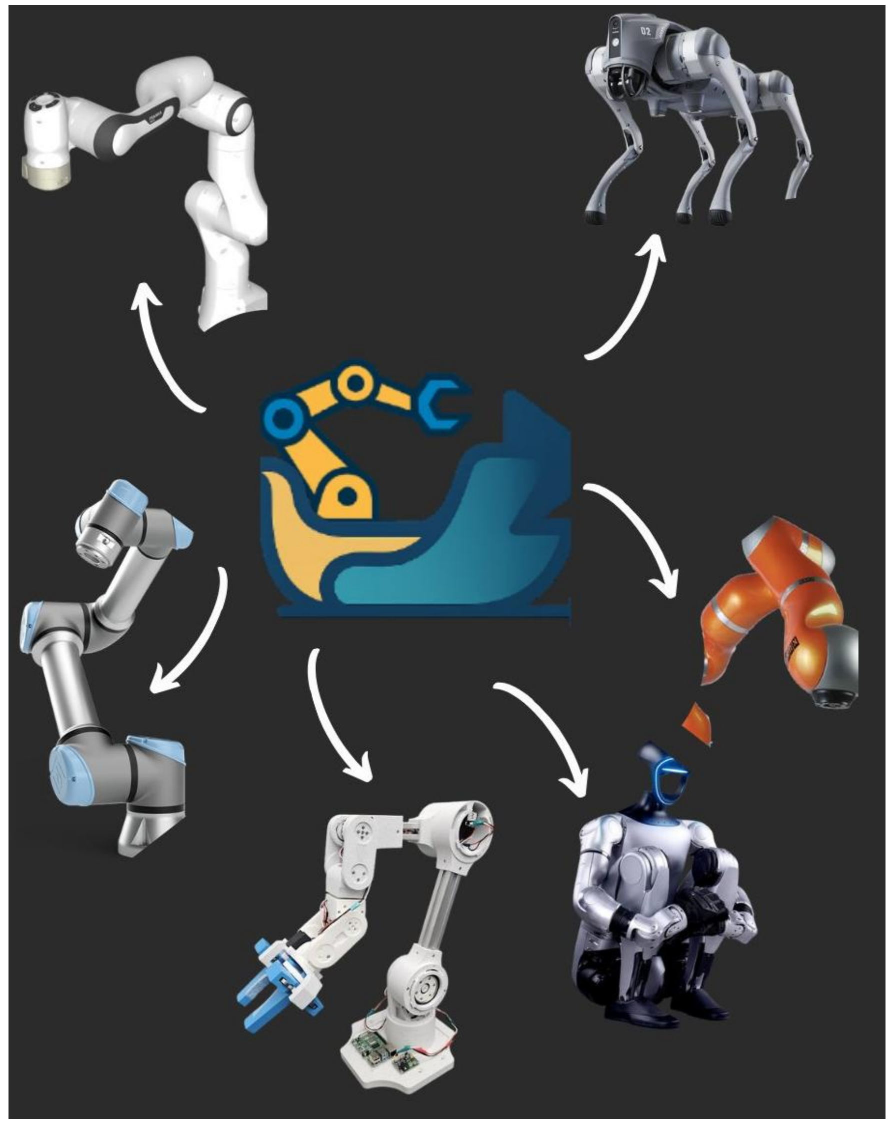
University of Windsor
School of Computer Science

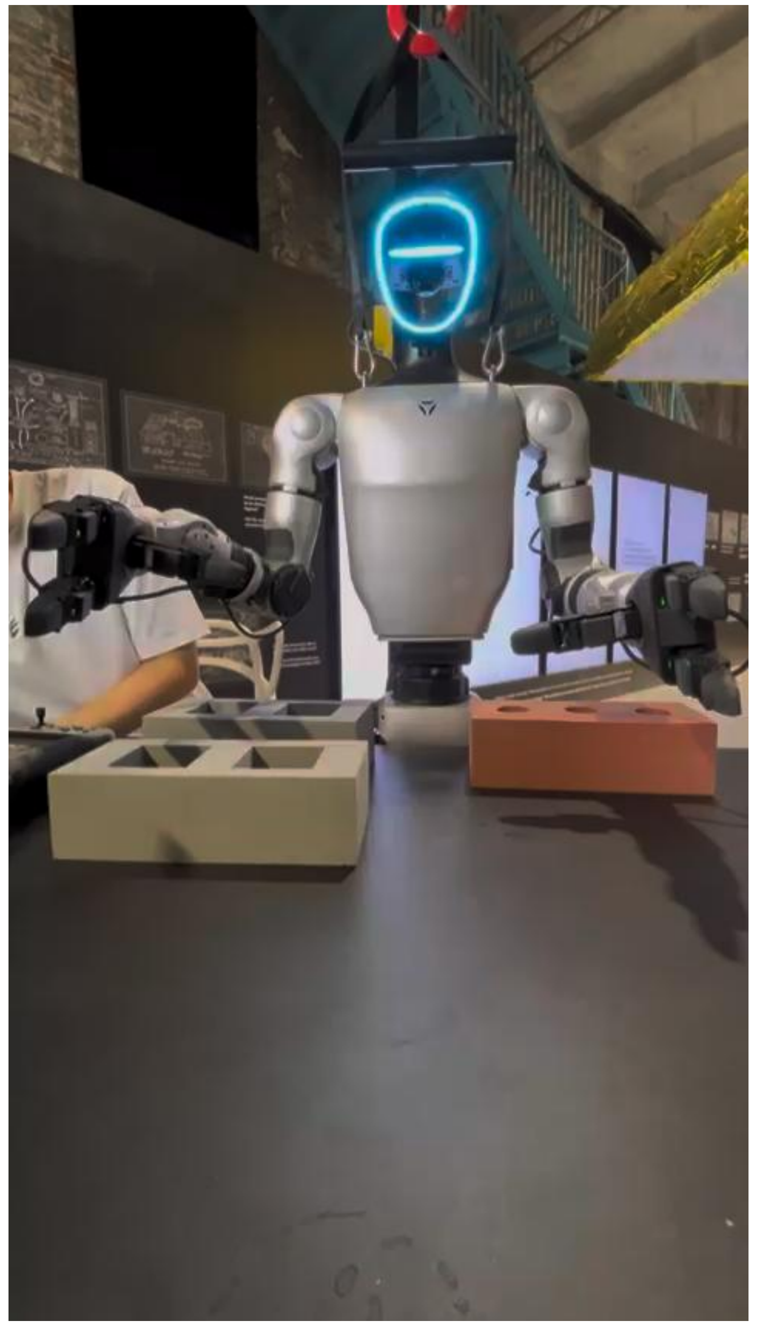
Ark – Future

- Young but growing.
- Expanding native support.
  - Their own printable arm!









University of Windsor
School of Computer Science

Future – University of Windsor

- Piloting a robotics course.
  - Computer science perspective.
    - Simulation and AI first.
  - Hope to grow into a specialization.
- Variety of robots.
  - Port to Ark.
  - Yahboom
    - DOFBOT – category.yahboom.net/products/dofbot-jetson_nano
    - Jetbot – category.yahboom.net/products/jetbot
    - RDK X3 – category.yahboom.net/collections/r-omnidirection/products/rdk-x3-robot
  - Build the Ark Bot – github.com/Robotics-Ark/ark_bot

University of Windsor
School of Computer Science

Thank You for Listening!

StevenRice.ca

Contact@StevenRice.ca

StevenRice99

StevenRice99

University of Windsor
School of Computer Science