Robot Learning: A Tutorial

Francesco Capuano **Caroline Pascal** **Adil Zouitine** **Thomas Wolf** **Michel Aractingi**

University of Oxford, Hugging Face

Abstract

Robot learning is at an inflection point, driven by rapid advancements in machine learning and the growing availability of large-scale robotics data. This shift from classical, model-based methods to data-driven, learning-based paradigms is unlocking unprecedented capabilities in autonomous systems. This tutorial navigates the landscape of modern robot learning, charting a course from the foundational principles of Reinforcement Learning and Behavioral Cloning to generalist, language-conditioned models capable of operating across diverse tasks and even robot embodiments. This work is intended as a guide for researchers and practitioners, and our goal is to equip the reader with the conceptual understanding and practical tools necessary to contribute to developments in robot learning, with ready-to-use examples implemented in `lerobot`.

Code: `https://github.com/huggingface/lerobot`
Date: **October 13, 2025**

Contents

Foreword

Robotics is an inherently multidisciplinary field, and is not witnessing unprecedented advancements since its inception in the 1960s. Yet, more than sixty years after the debut of Unimate, robots have still not fully integrated into the rich, unstructured, and dynamic world we humans inhabit. Over the decades, numerous disciplines have shown immense promise in tackling the challenges of creating autonomous systems. This tutorial takes a clear stance in the debate on whether modern Machine Learning can play a pivotal role in the development of autonomous robot systems: we believe this to be the case.

Nonetheless, we also hold that the wealth of research from both academia and industry in classical robotics over the past six decades is, simply put, too valuable to be cast aside in favor of purely learning-based methods. However, the interplay between classical robotics and modern machine learning is still in its nascent stages, and the path to integration yet to be clearly defined. In turn our goal here is to present what we consider to be the most relevant approaches within robot learning today, while warmly extending an invite to collaborate to expand the breadth of this work! Start contributing today here.

This tutorial...

- Does *not* aim to be a comprehensive guide to general field of robotics, manipulation or underactuated systems: Siciliano and Khatib (2016) and Tedrake (a,b) do this better than we ever could.
- Does *not* aim to be an introduction to statistical or deep learning: Shalev-Shwartz and Ben-David (2014) and Prince (2023) cover these subjects better than we ever could.

- Does *not* aim to be a deep dive into Reinforcement Learning, Diffusion Models, or Flow Matching: invaluable works such as Sutton and Barto (2018), Nakkiran et al. (2024), and Lipman et al. (2024) do this better than we ever could.

Instead, our goal here is to provide an intuitive explanation as per why these disparate ideas have converged to form the exciting field of modern robot learning, driving the unprecedented progress we see today. In this spirit, we follow the adage: "a jack of all trades is a master of none, *but oftentimes better than a master of one*."

We sincerely hope this tutorial serves as a valuable starting point for your journey into robot learning.

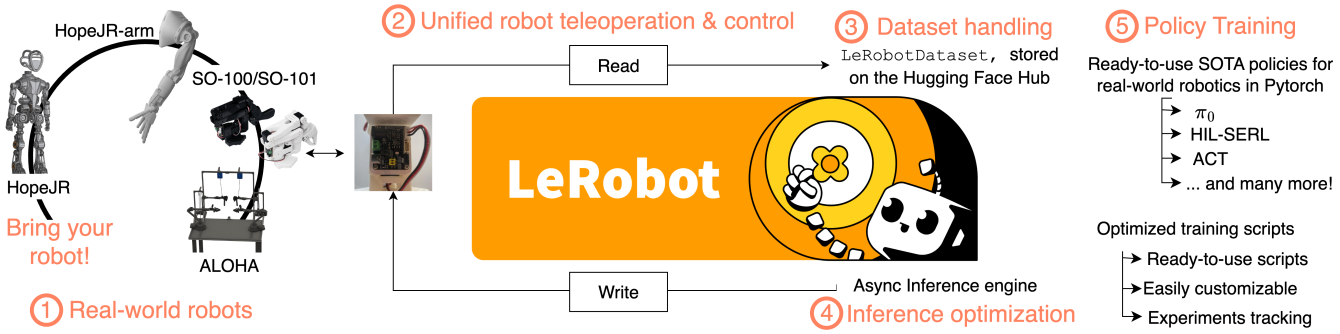

Figure 1 | `lerobot` is the open-source library for end-to-end robotics developed by Hugging Face. The library is vertically integrated on the entire robotics stack, supporting low-level control of real-world robot devices, advanced data and inference optimizations, as well as SOTA robot learning methods with simple implementations in pure Pytorch.

1 Introduction

Autonomous robotics holds the premise of relieving humans from repetitive, tiring or dangerous manual tasks. Consequently, the field of robotics has been widely studied since its first inception in the 1950s. Lately, advancements in Machine Learning (ML) have sparked the development of a relatively new class of methods used to tackle robotics problems, leveraging large amounts of data and computation rather than human expertise and modeling skills to develop autonomous systems.

The frontier of robotics research is indeed increasingly moving away from classical model-based control paradigm, embracing the advancements made in ML, aiming to unlock (1) monolithic perception-to-action control pipelines and (2) multi-modal data-driven feature extraction strategies, together with (3) reduced reliance on precise models of the world and (4) a better positioning to benefit from the growing availability of open robotics data. While central problems in manipulation, locomotion and whole-body control demand knowledge of rigid-body dynamics, contact modeling, planning under uncertainty, recent results seem to indicate learning can prove just as effective as explicit modeling, sparking interest in the field of *robot learning*. This interest can be largely justified considering the significant challenges related to deriving accurate models of robot-environment interactions.

Moreover, since end-to-end learning on ever-growing collections of text and image data has historically been at the core of the development of *foundation models* capable of semantic reasoning across multiple modalities (images, text, audio, etc.), deriving robotics methods grounded in learning appears particularly consequential, especially as the number of openly available datasets continues to grow.

Robotics is, at its core, an inherently multidisciplinary field, requiring a wide range of expertise in both *software* and *hardware*. The integration of learning-based techniques further broadens this spectrum of skills, raising the bar for both research and practical applications. `lerobot` is an open-source library designed to integrate end-to-end with the entire robotics stack. With a strong focus on accessible, real-world robots (1) `lerobot` supports many, openly available, robotic platforms for manipulation, locomotion and even whole-body control. `lerobot`also implements a (2) unified, low-level approach to reading/writing robot configurations to extend support for other robot platforms with relatively low effort. The library introduces `LeRobotDataset`, (3) a native robotics dataset's format currently being used by the community to efficiently record and share datasets. `lerobot` also supports many state-of-the-art (SOTA) algorithms in robot learning—mainly based on Reinforcement Learning (RL) and Behavioral Cloning (BC) techniques—with efficient implementations in Pytorch, and extended support to experimentation and experiments tracking. Lastly, `lerobot` defines a custom, optimized inference stack for robotic policies decoupling action planning from action execution, proving effective in guaranteeing more adaptability at runtime.

This tutorial serves the double purpose of providing useful references for the Science behind—and practical use of—common robot learning techniques. To this aim, we strike to provide a rigorous yet concise overview of the core concepts behind the techniques presented, paired with practical examples of how to use such techniques concretely, with code examples in `lerobot`, for researchers and practitioners interested in the field of robot learning. This tutorial is structured as follows:

- Section **??** reviews classical robotics foundations, introducing the limitations of dynamics-based approaches to robotics.

- Section ?? elaborates on the limitations of dynamics-based methods, and introduce RL as a practical approach to solve robotics problems, considering its upsides and potential limitations.
- Section ?? further describes robot learning techniques that aim at solving single-tasks learning, leveraging BC techniques to autonomously reproduce specific expert demonstrations.
- Section ?? presents recent contributions on developing generalist models for robotics applications, by learning from large corpora of multi-task & multi-robot data (*robotics foundation models*).

Our goal with this tutorial is to provide an intuitive explanation of the reasons various disparate ideas from Machine Learning (ML) have converged and are powering the current evolution of Robotics, driving the unprecedented progress we see today. We complement our presentation of the most common and recent approaches in robot learning with practical code implementations using `lerobot`, and start here by presenting the dataset format introduced with `lerobot`.

1.1 `LeRobotDataset`

`LeRobotDataset` is one of the most impactful features of `lerobot`, developed in keeping with the observation that robotics data is increasingly central in robot learning. Thus, `lerobot` defines a standardized dataset format designed to address the specific needs of robot learning research, providing a unified and convenient access to robotics data across modalities, including sensorimotor readings, multiple camera feeds and teleoperation status. `LeRobotDataset` also accommodates for storing general information regarding the data being collected, including textual descriptions of the task being performed by the teleoperator, the kind of robot used, and relevant measurement specifics like the frames per second at which the recording of both image and robot state's streams are proceeding.

In this, `LeRobotDataset` provides a unified interface for handling multi-modal, time-series data, and it is designed to seamlessly integrate with the PyTorch and Hugging Face ecosystems. `LeRobotDataset` can be easily extended by users and it is highly customizable by users, and it already supports openly available data coming from a variety of embodiments supported in `lerobot`, ranging from manipulator platforms like the SO-100 arm and ALOHA-2 setup, to real-world humanoid arm and hands, as well as entirely simulation-based datasets, and self-driving cars. This dataset format is built to be both efficient for training and flexible enough to accommodate the diverse data types encountered in robotics, while promoting reproducibility and ease of use for users.

1.1.1 The dataset class design

A core design choice behind `LeRobotDataset` is separating the underlying data storage from the user-facing API. This allows for efficient storage while presenting the data in an intuitive, ready-to-use format.

Datasets are always organized into three main components:

- **Tabular Data**: Low-dimensional, high-frequency data such as joint states, and actions are stored in efficient memory-mapped files, and typically offloaded to the more mature `datasets` library by Hugging Face, providing fast with limited memory consumption.
- **Visual Data**: To handle large volumes of camera data, frames are concatenated and encoded into MP4 files. Frames from the same episode are always grouped together into the same video, and multiple videos are grouped together by camera. To reduce stress on the file system, groups of videos for the same camera view are also broke into multiple sub-directories, after a given threshold number.
- **Metadata** A collection of JSON files which describes the dataset's structure in terms of its metadata, serving as the relational counterpart to both the tabular and visual dimensions of data. Metadata include the different feature schema, frame rates, normalization statistics, and episode boundaries.

For scalability, and to support datasets with potentially millions of trajectories (resulting in hundreds of millions or billions of individual camera frames), we merge data from different episodes into the same high-level structure. Concretely, this means that any given tabular collection and video will not typically contain information about one episode only, but rather a concatenation of the information available in multiple episodes. This keeps the pressure on the file system limited, both locally and on remote storage providers like Hugging Face, though at the expense of leveraging more heavily relational-like, metadata parts of the dataset, which are used to reconstruct information such as at which position, in a given file, an episode starts or ends. An example struture for a given `LeRobotDataset` would appear as follows:

- `meta/info.json`: This metadata is a central metadata file. It contains the complete dataset schema, defining all

features (e.g., `observation.state`, `action`), their shapes, and data types. It also stores crucial information like the dataset's frames-per-second (`fps`), `lerobot`'s version at the time of capture, and the path templates used to locate data and video files.

- `meta/stats.json`: This file stores aggregated statistics (mean, std, min, max) for each feature across the entire dataset, used for data normalization for most policy models and accessible externally via `dataset.meta.stats`.
- `meta/tasks.jsonl`: This file contains the mapping from natural language task descriptions to integer task indices, which are useful for task-conditioned policy training.
- `meta/episodes/*` This directory contains metadata about each individual episode, such as its length, the corresponding task, and pointers to where its data is stored in the dataset's files. For scalability, this information is stored in files rather than a single large JSON file.
- `data/*`: Contains the core frame-by-frame tabular data, using parquet files to allow for fast, memory-mapped access. To improve performance and handle large datasets, data from multiple episodes are concatenated into larger files. These files are organized into chunked subdirectories to keep the size of directories manageable. A single file typically contains data for more than one single episode.
- `videos/*`: Contains the MP4 video files for all visual observation streams. Similar to the `data/` directory, the video footage from multiple episodes is concatenated into single MP4 files. This strategy significantly reduces the number of files in the dataset, which is more efficient for modern filesystems.

1.2 Code Example: Batching a (Streaming) Dataset

This section provides an overview of how to access datasets hosted on Hugging Face using the `LeRobotDataset` class. Every dataset on the Hugging Face Hub containing the three main pillars presented above (Tabular, Visual and relational Metadata), and can be assessed with a single instruction.

In practice, most reinforcement learning (RL) and behavioral cloning (BC) algorithms tend to operate on stack of observation and actions. For the sake of brevity, we will refer to joint spaces, and camera frames with the single term of *frame*. For instance, RL algorithms may use a history of previous frames $o_{t-H_o:t}$ to mitigate partial observability, and BC algorithms are in practice trained to regress chunks of multiple actions (a_{t+t+H_a}) rather than single controls. To accommodate for these specifics of robot learning training, `LeRobotDataset` provides a native windowing operation, whereby users can define the *seconds* of a given window (before and after) around any given frame, by using the `delta_timestemps` functionality. Unavailable frames are opportunely padded, and a padding mask is also returned to filter out the padded frames. Notably, this all happens within the `LeRobotDataset`, and is entirely transparent to higher level wrappers commonly used in training ML models such as `torch.utils.data.DataLoader`.

Conveniently, by using `LeRobotDataset` with a Pytorch `DataLoader` one can automatically collate the individual sample dictionaries from the dataset into a single dictionary of batched tensors for downstream training or inference. `LeRobotDataset` also natively supports streaming mode for datasets. Users can stream data of a large dataset hosted on the Hugging Face Hub, with a one-line change in their implementation. Streaming datasets supports high-performance batch processing (ca. 80-100 it/s, varying on connectivity) and high levels of frames randomization, key features for practical BC algorithms which otherwise may be slow or operating on highly non-i.i.d. data. This feature is designed to improve on accessibility so that large datasets can be processed by users without requiring large amounts of memory and storage.

Code 1: Batching a (Streaming) Dataset

```python
import torch
from lerobot.datasets.lerobot_dataset import LeRobotDataset
from lerobot.datasets.streaming_dataset import StreamingLeRobotDataset

delta_timestamps = {
    "observation.images.wrist_camera": [-0.2, -0.1, 0.0]  # 0.2, and 0.1 seconds *before* each frame
}

# Optionally, use StreamingLeRobotDataset to avoid downloading the dataset
dataset = LeRobotDataset(
    "lerobot/svla_so101_pickplace",
    delta_timestamps=delta_timestamps
)
```

```python

# Streams frames from the Hugging Face Hub without loading into memory
streaming_dataset = StreamingLeRobotDataset(
    "lerobot/svla_so101_pickplace",
    delta_timestamps=delta_timestamps
)

# Get the 100th frame in the dataset by
sample = dataset[100]
print(sample)
# {
# 'observation.state': tensor([...]),
# 'action': tensor([...]),
# 'observation.images.wrist_camera': tensor([3, C, H, W]), for delta timesteps
# ...
# }

batch_size=16
# wrap the dataset in a DataLoader to use process it batches for training purposes
data_loader = torch.utils.data.DataLoader(
    dataset,
    batch_size=batch_size
)

# Iterate over the DataLoader in a training loop
num_epochs = 1
device = "cuda" if torch.cuda.is_available() else "cpu"

for epoch in range(num_epochs):
    for batch in data_loader:
        # Move data to the appropriate device (e.g., GPU)
        observations = batch["observation.state"].to(device)
        actions = batch["action"].to(device)
        images = batch["observation.images.wrist_camera"].to(device)

        # Next, you can do amazing_model.forward(batch)
        ...
```

1.3 Code Example: Collecting Data

Code 2: Recording Expert Data

```python
"""
You can also use the CLI to record data. To see the required arguments, run:
lerobot-record --help
"""
from lerobot.cameras.opencv.configuration_opencv import OpenCVCameraConfig
from lerobot.datasets.lerobot_dataset import LeRobotDataset
from lerobot.datasets.utils import hw_to_dataset_features
from lerobot.robots.so100_follower import SO100Follower, SO100FollowerConfig
from lerobot.teleoperators.so100_leader.config_so100_leader import SO100LeaderConfig
from lerobot.teleoperators.so100_leader.so100_leader import SO100Leader
from lerobot.utils.control_utils import init_keyboard_listener
from lerobot.utils.utils import log_say
from lerobot.utils.visualization_utils import init_rerun
from lerobot.scripts.lerobot_record import record_loop

NUM_EPISODES = 5
FPS = 30
EPISODE_TIME_SEC = 60
RESET_TIME_SEC = 10
TASK_DESCRIPTION = ...  # provide a task description

HF_USER = ...  # provide your Hugging Face username

follower_port = ...  # find your ports running: lerobot-find-port
leader_port = ...
```

```
follower_id = ...  # to load the calibration file
leader_id = ...

# Create the robot and teleoperator configurations
camera_config = {"front": OpenCVCameraConfig(
    index_or_path=0, width=640, height=480, fps=FPS)
}
robot_config = SO100FollowerConfig(
    port=follower_port,
    id=follower_id,
    cameras=camera_config
)
teleop_config = SO100LeaderConfig(
    port=leader_port,
id=leader_id
)

# Initialize the robot and teleoperator
robot = SO100Follower(robot_config)
teleop = SO100Leader(teleop_config)

# Configure the dataset features
action_features = hw_to_dataset_features(robot.action_features, "action")
obs_features = hw_to_dataset_features(robot.observation_features, "observation")
dataset_features = {**action_features, **obs_features}

# Create the dataset where to store the data
dataset = LeRobotDataset.create(
    repo_id=f"{HF_USER}/robot-learning-tutorial-data",
    fps=FPS,
    features=dataset_features,
    robot_type=robot.name,
    use_videos=True,
    image_writer_threads=4,
)

# Initialize the keyboard listener and rerun visualization
_, events = init_keyboard_listener()
init_rerun(session_name="recording")

# Connect the robot and teleoperator
robot.connect()
teleop.connect()

episode_idx = 0
while episode_idx < NUM_EPISODES and not events["stop_recording"]:
    log_say(f"Recording episode {episode_idx + 1} of {NUM_EPISODES}")

    record_loop(
        robot=robot,
        events=events,
        fps=FPS,
        teleop=teleop,
        dataset=dataset,
        control_time_s=EPISODE_TIME_SEC,
        single_task=TASK_DESCRIPTION,
        display_data=True,
    )

    # Reset the environment if not stopping or re-recording
    if not events["stop_recording"] and (episode_idx < NUM_EPISODES - 1 or events["rerecord_episode"]):
        log_say("Reset the environment")
        record_loop(
            robot=robot,
            events=events,
            fps=FPS,
            teleop=teleop,
            control_time_s=RESET_TIME_SEC,
            single_task=TASK_DESCRIPTION,
            display_data=True,
        )

```

```
    if events["rerecord_episode"]:
        log_say("Re-recording episode")
        events["rerecord_episode"] = False
        events["exit_early"] = False
        dataset.clear_episode_buffer()
        continue

    dataset.save_episode()
    episode_idx += 1

# Clean up
log_say("Stop recording")
robot.disconnect()
teleop.disconnect()
dataset.push_to_hub()
```

References

Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky T. Q. Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. Flow Matching Guide and Code, December 2024.

Preetum Nakkiran, Arwen Bradley, Hattie Zhou, and Madhu Advani. Step-by-Step Diffusion: An Elementary Tutorial, June 2024.

Simon J.D. Prince. *Understanding Deep Learning*. The MIT Press, 2023.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 1 edition, May 2014. ISBN 978-1-107-05713-5 978-1-107-29801-9. doi: 10.1017/CBO9781107298019.

Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer Handbooks. Springer International Publishing, Cham, 2016. ISBN 978-3-319-32550-7 978-3-319-32552-1. doi: 10.1007/978-3-319-32552-1.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

Russ Tedrake. Robotic Manipulation. Perception, Planning and Control., a.

Russ Tedrake. Underactuated Robotics. Algorithms for Walking, Running, Swimming, Flying, and Manipulation, b.