?????? ? ????????

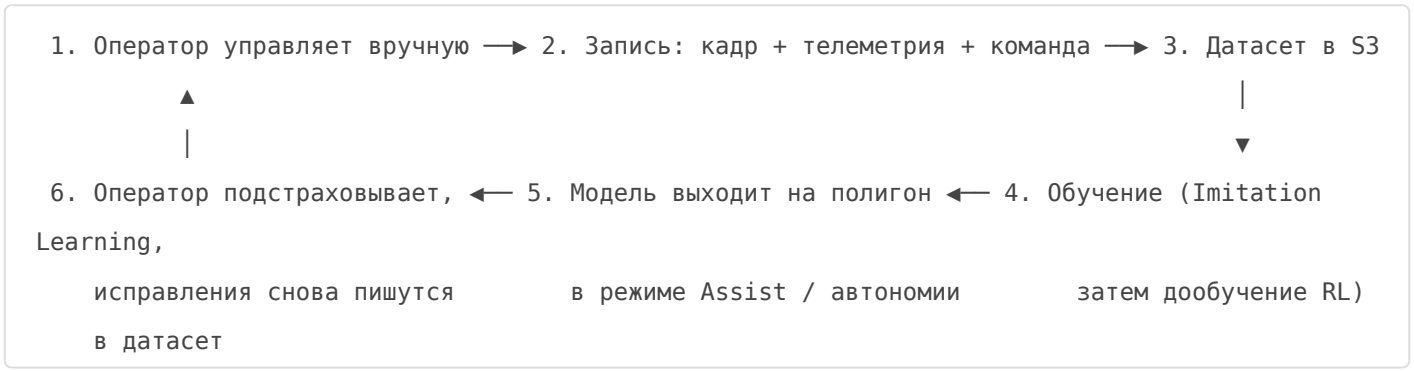
Сбор датасетов, Imitation Learning, RL, полигоны

- Сбор данных и обучение моделей
- Физический и виртуальный полигоны

???? ?????? ? ????????
???????

Каждая сессия ручного управления на платформе — это готовые обучающие данные: платформа синхронно записывает, **что видел робот** (видео), **в каком он был состоянии** (телеметрия) и **что сделал оператор** (команды). На этих данных обучается следующая версия модели.

???? ????????? ??????



Особенно ценны **вмешательства оператора** в работу модели: момент, когда человек перехватил управление, показывает, где модель ошибается. Такие фрагменты стоит брать в датасет с повышенным весом (подход DAgger).

?????? ??????

Поле	Источник	Пример
`t`	Время записи	`1790155251.120`
`robot`	Идентификатор робота	`tank-01`
`frame`	Ссылка на кадр в S3	`s3://artex-video/tank-01/s42/rgb/1790155251120.jpg`
`telemetry`	Актуальный вектор телеметрии	`{"battery": 11.8, "speed": 0.4}`
`command`	Последняя команда	`forward`
`author`	Автор команды	`operator-7` или `model:demo-rules@1.0`

?????????? ????????

1. **Синхронизация.** Каждому кадру сопоставьте последнюю команду, отданную до момента кадра (с учётом задержки — обычно сдвиг на 100–200 мс вперёд даёт лучшие результаты).
2. **Фильтрация.** Удалите участки, где робот стоял без команд, и участки с потерей связи.
3. **Балансировка.** Команды `forward` обычно преобладают — уравновесьте классы (прореживание или веса).
4. **Разбиение.** Делите на обучающую и проверочную выборки **по сессиям**, а не по кадрам, иначе соседние почти одинаковые кадры попадут в обе выборки.

```
import pandas as pd

log = pd.read_parquet("session_42.parquet")          # t, frame, telemetry, command, author
log = log.sort_values("t")
log["command"] = log["command"].ffill()              # команда действует до следующей
log = log[log["command"].notna()]
log = log[~((log["command"] == "stop") & (log["command"].shift() == "stop"))]  # длинные
стоянки
print(log["command"].value_counts())
```

Imitation Learning (Behavior Cloning)

Модель учится предсказывать команду оператора по кадру (и телеметрии). Минимальный пример на PyTorch:

```
import torch, torch.nn as nn
from torchvision import models

COMMANDS = ["forward", "back", "left", "right", "stop"]

class Policy(nn.Module):
    def __init__(self):
        super().__init__()
        self.backbone = models.mobilenet_v3_small(weights="DEFAULT")
        self.backbone.classifier[-1] = nn.Linear(1024, len(COMMANDS))

    def forward(self, x):                # x: [B, 3, 224, 224]
```

```
        return self.backbone(x)

policy = Policy()
opt = torch.optim.AdamW(policy.parameters(), lr=3e-4)
loss_fn = nn.CrossEntropyLoss()

for images, labels in train_loader:       # DataLoader по подготовленному датасету
    opt.zero_grad()
    loss = loss_fn(policy(images), labels)
    loss.backward()
    opt.step()
```

Обученная модель подключается к роботу так же, как демо-модель на странице Разработка модели ИИ для платформы: метод `step()` возвращает `COMMANDS[argmax]`.

???????? ? ????????????? (RL)

Behavior Cloning ограничен качеством операторов. Чтобы модель превзошла человека, её дообучают методами RL — как правило, на виртуальном полигоне, где можно проводить тысячи эпизодов без риска для техники:

- **Наблюдение** — кадр (или признаки из YOLO) + телеметрия;
- **Действие** — команда из словаря;
- **Награда** — задаётся задачей: продвижение по трассе, выполнение задания, победа в поединке; штрафы за столкновения и выезд за границы;
- **Инициализация** — весами модели, обученной Imitation Learning, что сильно ускоряет обучение.

?????????????? ???????

Храните каждую версию модели в S3 вместе с описанием: на каких сессиях обучена, метрики на проверочной выборке, результаты на полигоне (в том числе ELO в соревнованиях). Автор команд модели должен содержать имя и версию (`model:<name>@<version>`) — так в записях видно, какая версия что делала.

?????????? ? ??????????? ????????

Платформа ARTEX объединяет два мира: физический полигон с реальными устройствами и виртуальный полигон с симуляцией. Оба подключаются к платформе одинаково, поэтому модели и интерфейс работают с ними без изменений.

?????????

	Физический полигон	**Виртуальный полигон**
Устройства	RC-танки, машинки, дроны на реальной трассе	Модели роботов в симуляторе (Unity / Gazebo)
Видео	FPV-камера на каждом устройстве	Рендер камеры симулятора
Телеметрия	Скорость, GPS, батарея, датчики	Те же поля, вычисленные симулятором
Подключение	SDK `artex-robot`	SDK `artex-robot` — тот же протокол
Для чего	Соревнования, итоговая проверка, сбор реальных данных	Прототипирование ИИ, тысячи итераций RL без риска для оборудования

??????????? ?????????? ? ?????????

Каждый робот в симуляции — отдельный клиент SDK. Мост между симулятором и SDK:

1. Получает из симулятора кадр виртуальной камеры и состояние робота.
2. Передаёт их через `@robot.video` и `@robot.telemetry`.
3. Команды из `@robot.com_handler` передаёт в симулятор как управляющие воздействия.

Простейший пример самодостаточного виртуального робота — на странице Примеры роботов (пример 3). Для Gazebo мост удобно реализовать как узел ROS2, для Unity — через TCP/UDP-соединение с Python-скриптом моста.

>

> Чтобы модель переносилась на реальное устройство без изменений, виртуальный робот должен совпадать с реальным по словарю команд, набору и единицам измерения телеметрии, разрешению и положению камеры.

??????? ?? ????????? ? ??????????? (sim2real)

Модель, хорошо работающая в симуляции, часто хуже работает на реальном роботе из-за различий в изображении и физике. Рекомендации:

- **Рандомизация домена** — в симуляции меняйте освещение, текстуры, цвета, шум камеры, трение и задержки.
- **Реалистичные задержки** — добавьте в симулятор задержку видео и команд, как на реальной сети (100–200 мс).
- **Промежуточное представление** — обучайте политику на признаках (детекции YOLO, сегментация), а не на «сырых» пикселях: признаки меньше отличаются между симуляцией и реальностью.
- **Дообучение на реальных данных** — после переноса соберите реальные сессии с подстраховкой оператора и дообучите модель (см. Сбор данных и обучение).
- **Безопасный первый запуск** — ограничьте скорость, держите оператора наготове с кнопкой «Стоп».