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

Модели ИИ, протоколы управления, потоки данных: веб-интерфейс оператора, режимы управления и состязаний, разработка моделей, конвейеры модулей, сбор данных и обучение.

- Введение
  - Управление роботами с помощью ИИ: обзор
  - Режимы управления и состязаний
- Интерфейс оператора
  - Веб-интерфейс управления роботами
- Разработка моделей ИИ
  - Разработка модели ИИ для платформы
  - Конвейеры модулей
- Данные и обучение
  - Сбор данных и обучение моделей
  - Физический и виртуальный полигоны

????????

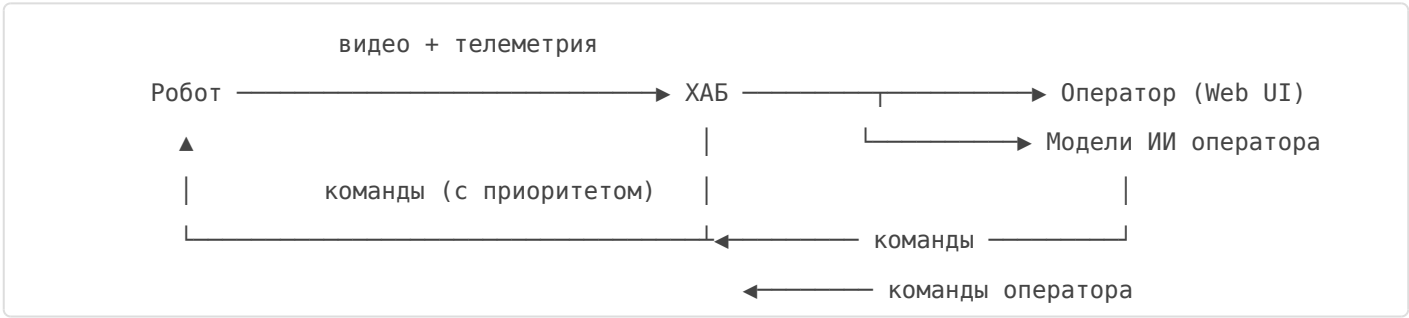
Как ИИ управляет роботами и режимы состязаний

Введение

?????????? ???????? ? ??????? ??: ?????

Платформа ARTEX позволяет управлять роботом человеку, модели ИИ или обоим одновременно. Модели ИИ работают в распределённой вычислительной среде платформы (или на борту робота), получают от ХАБа видео и телеметрию и возвращают команды управления — в том же формате, что и оператор.

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



- **Основной источник команд — модель ИИ.** Оператор управляет роботом при обучении модели или для «подстраховки» (перехватывает управление в опасной ситуации).
- **Одна команда — один формат.** Для робота нет разницы, кто отправил команду: различается только автор (`author`), по которому ХАБ применяет приоритет.
- **Каждый оператор может подключить свои модели** (Модели ИИ оператора 1 … N) — они запускаются в контейнерах вычислительной среды.

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

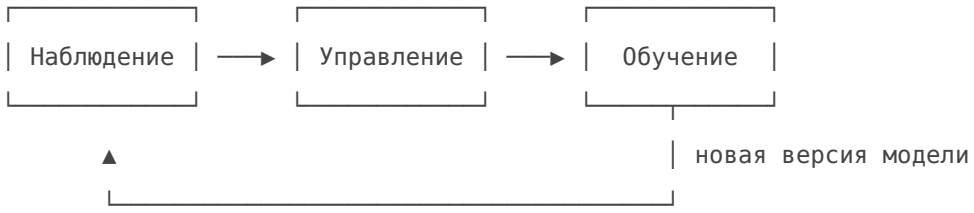
Уровень	Кто управляет	Роль ИИ	Роль оператора
Ручное управление	Оператор	Не участвует или только анализирует (детекция объектов поверх видео)	Полное управление: джойстик, клавиатура, VR-контроллер

Уровень	Кто управляет	Роль ИИ	Роль оператора
Assist (гибридный режим)	Оператор + ИИ	«Второй пилот»: подсказки, стабилизация, удержание курса, автоматическая остановка перед препятствием	Задаёт цель и основное направление
Полная автономия	ИИ	Принимает все решения по видео и телеметрии	Наблюдает, может в любой момент вмешаться (приоритет оператора)

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

Платформа поддерживает полный цикл: от наблюдения и ручного управления до сбора датасета и обучения следующей модели.

Этап	Технологии	Что происходит
Наблюдение	WebRTC + WebSocket	FPV-видео; телеметрия в JSON (GPS, скорость, батарея, IMU); дашборд реального времени
Управление	WebSocket-команды	Ручной режим → Assist → полная автономия ИИ; джойстик / клавиатура / VR-контроллер
Обучение	Запись в S3	Каждое действие оператора записывается в датасет; Imitation Learning для обучения следующей модели



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

Модель	Задача	Входы	Выходы
YOLO	Детекция объектов на видео в реальном времени	Кадры	Список объектов: класс, уверенность, рамка

Модель	Задача	Входы	Выходы
Assist AI	Помощь оператору (co-pilot)	Кадры, телеметрия, команды оператора	Скорректированные команды, подсказки
RL Agent	Управление на основе обучения с подкреплением	Признаки из видео, телеметрия	Команды управления
Custom	Любые пользовательские модели	Любые данные ХАБа	Команды, признаки, события

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

Вариант	Когда выбирать	Пример
В вычислительной среде платформы (Docker, K8s, GPU)	Тяжёлые модели, соревнования ИИ vs ИИ, централизованный контроль	YOLO на сервере, RL-агент
В конвейере модулей сервера	Типовая обработка, собираемая без программирования	YoloModule в визуальном редакторе
На борту робота	Нужна минимальная задержка или работа при плохой связи	Лёгкая детекция на Jetson / Raspberry Pi; результаты передаются как признаки в телеметрии

? ???? ??????

1. Познакомьтесь с веб-интерфейсом управления роботами.
2. Выберите режим управления или состязаний.
3. Разработайте модель по руководству Разработка модели ИИ для платформы.
4. Соберите конвейер обработки в визуальном редакторе — Конвейеры модулей.
5. Запишите данные и обучите следующую версию — Сбор данных и обучение.

Введение

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

Битва интеллектуальных роботов — соревновательный формат платформы ARTEX. Роботы на арене (физической или виртуальной) управляются людьми, моделями ИИ или их комбинацией. Побеждает не тот, у кого мощнее железо, а тот, у кого умнее алгоритм.

??????

Режим	Кто управляет	Суть	Технологии
Человек vs Человек	Операторы вручную	Классический формат: операторы управляют роботами с джойстика или VR-контроллера по FPV-видео	WebSocket + FPV
Человек vs ИИ	Оператор против автономной модели	Тест навыков человека против алгоритма; ИИ может помогать оператору (Assist)	YOLO + Assist AI
ИИ vs ИИ	Только модели	Полностью автономные модели: алгоритм против алгоритма. Побеждает код, а не оператор	RL + Socket API
Соревнования и турниры	Команды	ELO-рейтинг команд; школьные, кадетские, вузовские лиги; публичный лидерборд	ELO + Leaderboard

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

??????????

1. **Регистрация команд** — администратор создаёт учётные записи операторов и выдаёт им видимость своих роботов.
2. **Подключение роботов** — каждый робот подключается через SDK `artex-robot` и проверяется: видео, телеметрия, команды, кнопка «Стоп».

3. **Загрузка моделей** (для режимов с ИИ) — команды загружают модели в вычислительную среду; модели проходят тестовый заезд.
4. **Правила** — задаются: длительность раунда, критерии победы, ограничения скорости, допустимые команды.

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

1. Судья запускает раунд; включается запись всех потоков (видео, телеметрия, команды).
2. Операторы и/или модели управляют роботами.
3. Судья может остановить любого робота (Стоп / DISARM).
4. Результат раунда фиксируется и влияет на рейтинг.

????? ??????

- Запись раунда доступна для разбора: видео синхронизировано с телеметрией и командами.
- Данные раундов «Человек vs Человек» — датасет для обучения моделей следующего поколения.

??????? ELO

После каждого поединка рейтинг участников A и B пересчитывается:

```
E_A = 1 / (1 + 10^((R_B − R_A) / 400))        # ожидаемый результат A
R_A' = R_A + K · (S_A − E_A)                   # S_A = 1 победа, 0.5 ничья, 0 поражение
```

Рекомендуемые параметры: начальный рейтинг 1000, `K = 32` для новых участников (менее 10 поединков) и `K = 16` для остальных.

```
def elo_update(r_a, r_b, score_a, k=32):
    expected_a = 1 / (1 + 10 ** ((r_b - r_a) / 400))
    new_a = r_a + k * (score_a - expected_a)
    new_b = r_b + k * ((1 - score_a) - (1 - expected_a))
    return round(new_a), round(new_b)

elo_update(1000, 1100, 1)   # → (1020, 1080): победа над более сильным даёт больше очков
```

Рейтинги ведутся отдельно для людей, моделей и смешанных команд.

????

Лига	Участники	Особенности
Школьная	Школьники, кружки робототехники	Режимы «Человек vs Человек» и «Человек vs ИИ», готовые модели, геймификация
Кадетская	Кадетские корпуса, ведомственные центры	Управление БПЛА, тактические сценарии
Вузовская	Студенты, аспиранты, лаборатории	Режим «ИИ vs ИИ», собственные модели, публикация результатов

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

Веб-интерфейс и визуальный редактор модулей

Интерфейс оператора

???-?????????? ??????????? ????????

??? ????? ARTEX framework

ARTEX framework - ??? ??????????? ????????? ??? ?????????? ??????????, ??????????? ? ?????????? ??????????? ?????????: ??????? ???????????? ?????, ?????? ? ?????. ??? ????????????? ?????? ???-?????????, ????? ??????? ???????? ????? ?????? ????????? ???? ???????????? ?????????, ???????? ?? ??????? ? ???????? ?????????? ? ???????? ???????.

????????? ????????? ?? ?????????? ????????, ??? ????????? ????? ????????? ????? ???? ????????? ? ?????????????? ?????????. ARTEX ????????????? ??? ?? ???????? ?????????????, ??? ? ?? ?????????????: ?????? ???????? ??????? ????? ??????????, ?????? - ?????????????? ??? ???????? ??????????? ??????? ? ?????????? ????????????? ????????????.

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

A. ????? ?????? ???????????

????? ???????? ARTEX ???????????? ???????? ?? ???????? «???????????».



????? ???????????? ??????? ?????????? ??? ???? ???????? ???????????, ??????? ? ?????? ?????? ???????? ?????????.

???????? ??????????:

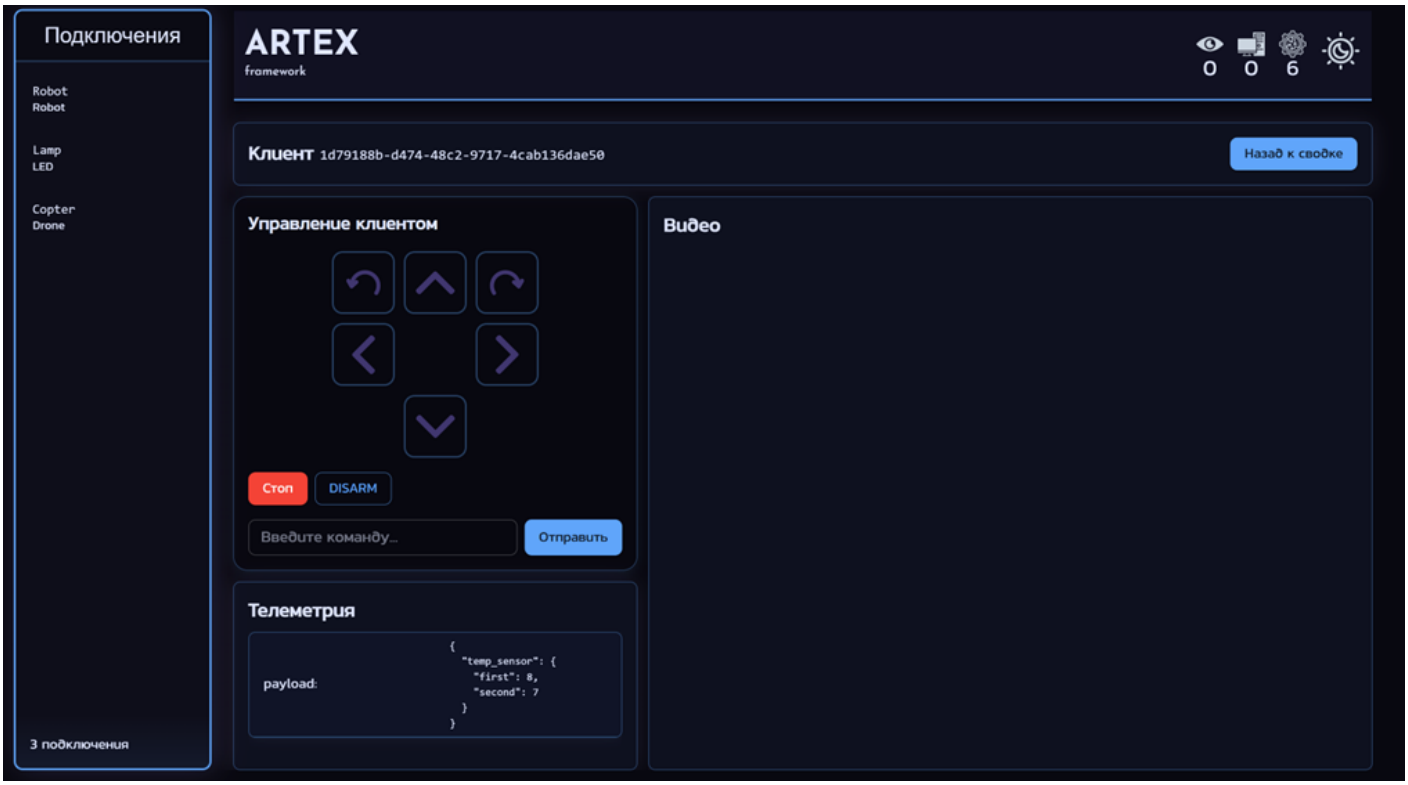
- **?????? ???????????** – ??????????? ? ?????????? ????? ????????? (Robot, LED, Drone). ????????? ?????? ???????, ????? ???????????? ????????? ? ????.
- **??????? ???????????** – ????????? ?????????? ?? ??????? ??????????:

    1. SID - ?????????? ????????????? ??????????.
    2. ??? - ???????????????? ???.
    3. Tun - ??????????? ???.
    4. BTK - ?????????? ???????.
    5. ???????????? - ???? ??? ??????? ???????? ???? ???????.

??????????? – ?????? ???????????? ?????????. ???? ?? ?????? ???????? ??????? ????????? ????????????? ???????? ?????????? ????????? ???????????.

B. ?????????? ???????????? Robot

??? ??????? ?? ??????? Robot ??????????? ????????? ?????????? ???????.



???? «??????» ???????? ?????????? ????????????? ??????????.

?????? ?????????? ????????:

- «????» - ?????????? ????????? ??????????.
- «DISARM» - ?????? ????????????? ??????? ?????.
- ???? ????? ?????? ????????? ?????????? ????????? ?????:

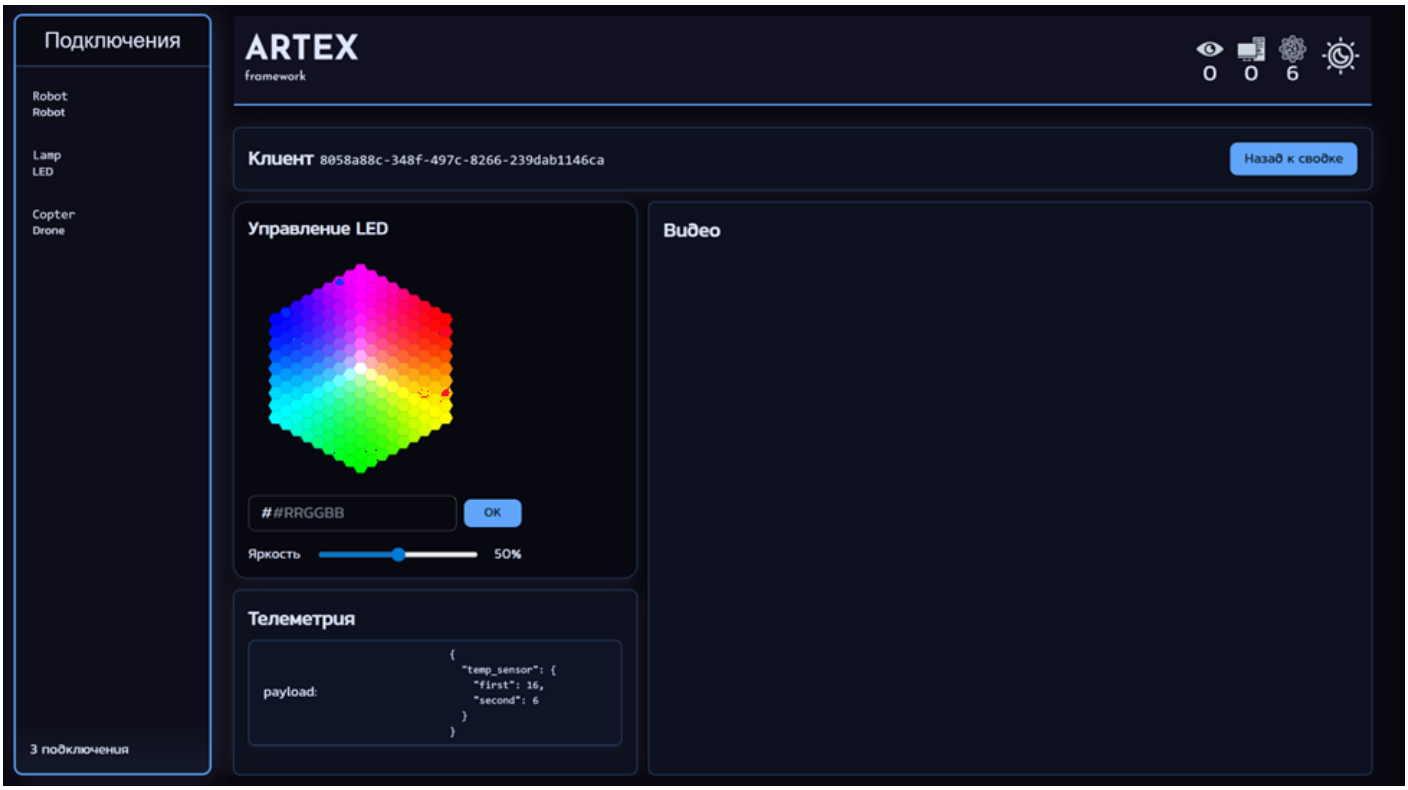
    1. video_start_record,
    video_stop_record – ?????? ????? ? ???? (?????????? ? ??????? ?????????? ?

?????? VideoRecordModule)

2. auto_run, auto_stop – ?????? ? ????????? ?????????????? ?????????? ?? ?????? ??????????

?????????? ? ??????? JSON ?????????? ????????? ???????? ? ???????? ???????. ??? ????????, ??? ?????????? ????? ????????????? ??????, ?????????? ??? ????????. ?????? JSON ??????????? ? ????????? ????? ????????? ????? ??????? ??? ????????? ??????????.

C. ?????????? ??????????? LED



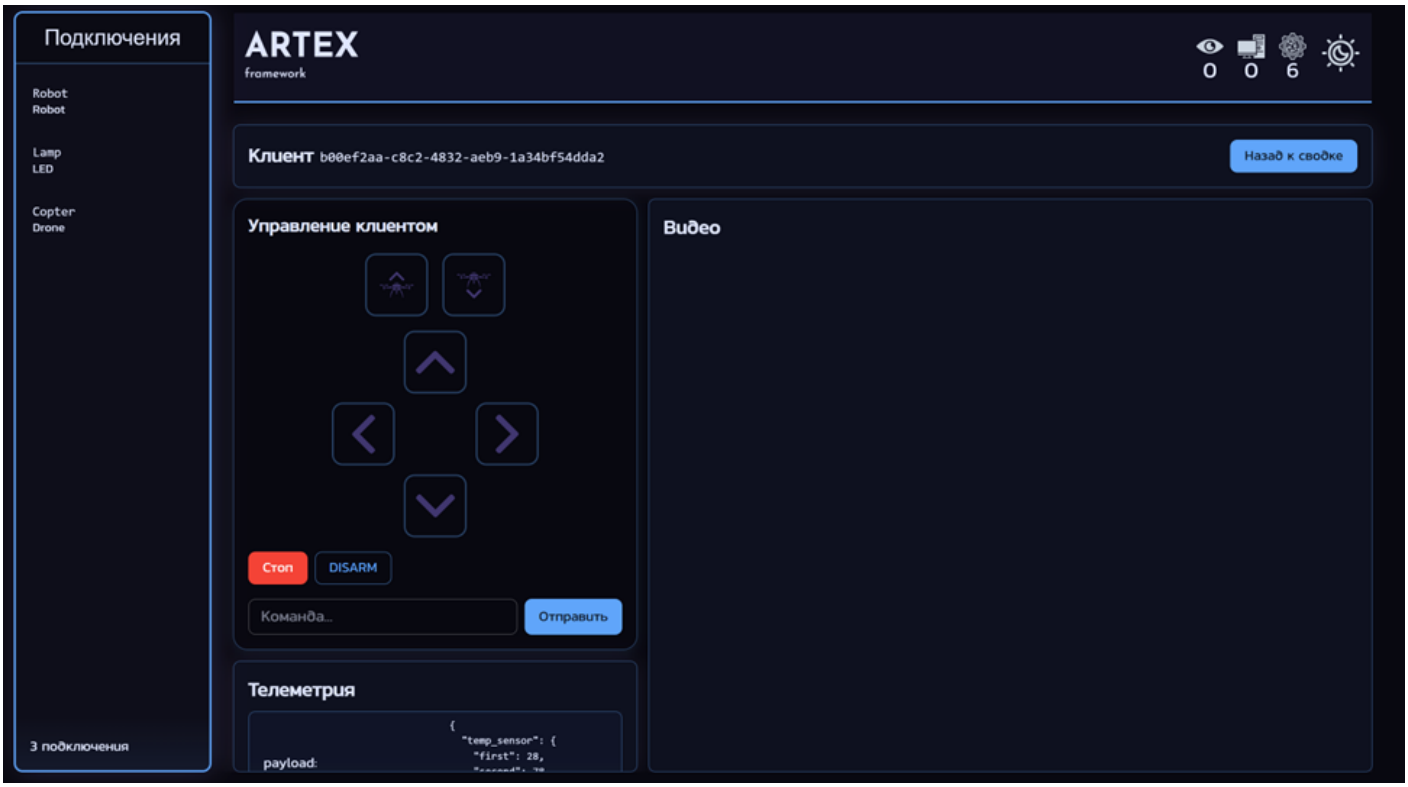
???? «??????» ????? ???????? ?????????? ????????????? ??????????.

?????? ?????????? ????????? ???????? ???? ?? ????????????, ? ????? ??????? ???????????.

- ???? ????? ????? ? ??????? **#RRGGBB** ? ?????? **«OK»** ??? ????? ?????.
- ???????? **«???????»** ? ???????????? ?? 0 ?? 100%.

?????????? ? ??????? JSON ?????????? ????????? ????????.

D. ?????????? ??????????? Drone


Подключения
Robot
Robot
Lamp
LED
Copter
Drone
3 подключения
ARTEX
framework
0 0 6
Клиент b00ef2aa-c8c2-4832-aeb9-1a34bf54dda2
Назад к сводке
Управление клиентом
Стоп
DISARM
Команда...
Отправить
Телеметрия
payload:
{
"temp_sensor": {
"first": 28,
Видео

???? «??????» ????? ???????? ?????????? ????????????? ??????????.

?????? ?????????? ????????:

- **«????»** – ?????????? ????????? ?????????? (????????? ??? ??????????? ????????).
- **«DISARM»** – ?????? ????????????? ??????? ????? (?????????? ????? ????? ???????).
- ???? ????? ???????????? ??????? ? ?????? **«?????????»**.

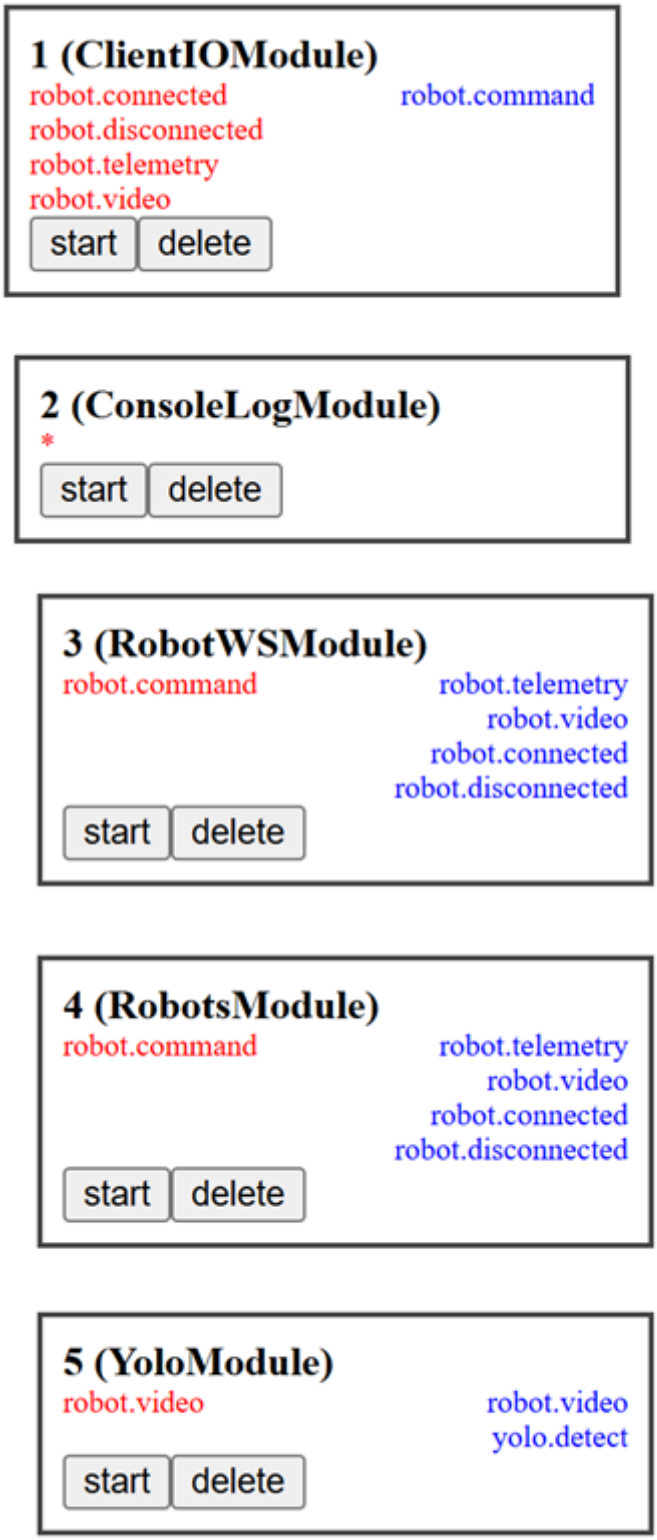
?????????? ? ??????? JSON ?????????? ????????? ???????? ? ???????? ???????.

?????????? ???????????: ?????? ? ?????? ??????

?? ??????? ???????????????? ??????????? ?????????? ?????? ????????? ???????. ??? ????????? ????????????? ????????????? ?????? ????????? ??????, ???????? ??????? ????????????? ?????-?????? ? ?????????? ?????????.

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

??? ??????????, ??????? ????????? ????????? ????????????? ????? (??????) ??? ???????? ?????? ????????? ??????. ?????? ?????? ???????? ?? ?????????? ?????? ? ????? ????? ? ??????. ???????????? ????????????? ?????? ?? ??????? ???? ? ????????? ?? ?????, ???????? ????????? ??????.



?????? 1: ClientIOModule

??????????: ?????? ???????? ? ???-??????????? ??? ?????????? ????????????.

?????:

- **robot.connected** - ??????? ??????????? ? ????????? ? ??????????? ??????.
- **robot.disconnected** - ??????? ?????? ?? ?????????? ??????.
- **robot.telemetry** - ???? ??? ???????????????? ??????.
- **robot.video** - ???? ??? ???????????.

???????? ?????????????: ??????????? ? ???????? RobotWSModule ??? RobotsModule, ????? ?????????? ?????????? ?? ????????? ?????? ?? ????? ?????????.

?????? 2: ConsoleLogModule

???????????: ????????? ?????? ??? ?????? ?????????? ?????????? ? ??????? ???????. ??????? ??? ?????????? ? ??????? ??????? ???????.

?????: ???????????. ?????? ?????? ???????? ??? ?????????? ????? ???? ??????, ?????????? ?? ????????? ?????????.

???????? ?????????????: ??????????? ?? ????? ??? ??????? – ????????? ?????? ? ???????, ????? ?????? ?????????? ????? ????????.

?????? 3: RobotWSModule

???????????: ???????? ?? ?????????????? ? ????????? ????????, ??????? ??? ??????? ??????????? ????? ???????? WebSocket (???????????? ???????????? ????? ? ???????? ???????).

?????:

- **robot.command** - ????? ??? ???????? ?????? ??????????.
- **robot.telemetry** - ???? ??? ?????? ?????????? ?? ?????????? (????? ????? ???????????? ? ClientIOModule).
- **robot.video** - ???? ??? ?????? ??????????? ?? ??????????.
- **robot.connected** - ???????? ??????, ???????????? ??? ??????????? ??????????.
- **robot.disconnected** - ???????? ?????? ??? ??????????.

???????? ?????????????: ???????? ?????? ????? ?????????? ??????????? ? ??????????. ???????? ?????? ?? ?????? ? ???????? ?? ? ?????? ?????? (????????, ClientIO ??? ???????????, Yolo ??? ?????????). ????? ????????? ??????? ?? ?????????? ? ?????????? ?? ??????.

?????? 4: YoloModule

???????????: ?????? ????????????? ?????? ?? ???? ????????? YOLO (You Only Look Once). ????????? ???????? ???????? ?? ???????????? ? ???????? ???????.

?????:

- **robot.video** – ???? ??? ??????????? (???????????? ? ?????? robot.video ?????? ??????, ???????? RobotWSModule).
- **yolo.detect** – ????? ? ???????????? ????????. ??? ?????? ????? ???? ???????? ? ClientIOModule ??? ??????????? ?????? ????? ??? ?????????.

???????? ?????????????: ??????????? ? ?????????? ??????????? (RobotWSModule ??? RobotsModule) ? ???????? ?????????? ???????? ? ????????? ??? ? ?????? ?????? ??? ???????????? ???????.

??????????

ARTEX framework - ??? ???????, ???????????? ??????????, ?????????? ? ?????????. ????????? ????????? ??????????? ??? ????? ???????????? ??? ????? ??????: ?? ?????? ???? ?? ???????????? ?????????????. ??????????? ????????? ??????????? ?????? ? ?????????? ????? ?????????? ?????? ARTEX ????????????? ????????, ??????? ? ???????? ?????? ? ????????????? ????? ?????????????.

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

Контракт модели, демо-модель, конвейеры модулей

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

Руководство для разработчиков моделей ИИ: как устроен контракт модели с платформой и как реализовать правила демо-прототипа.

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

Модель на платформе ARTEX — это функция «наблюдение → команда»:



Элемент	Формат	Источник
Кадр	numpy-массив BGR (после декодирования JPEG)	Поток видео робота, актуальный кадр из кэша ХАБа
Телеметрия	Словарь, например `{"battery": 11.8, "pose": {...}}`	Актуальный вектор телеметрии с временем создания
Команда	Строка или структура из словаря команд робота (см. «ПО для робота» → Обработка команд)	Отправляется роботу через ХАБ от имени модели
`None`	«Ничего не менять»	—

Требования к модели:

- **Работать с актуальными данными.** ХАБ отдаёт последний кадр и последний вектор телеметрии; модель не должна рассчитывать на получение каждого кадра.
- **Укладываться в бюджет задержки.** Суммарно «камера → решение → привод» — порядка 250 мс; на решение модели разумно закладывать не более 50–100 мс.
- **Не повторять одну и ту же команду без необходимости** — отправлять команду при смене решения (или с частотой, которую требует сторожевой таймер робота).

- **Учитывать приоритет оператора.** Если оператор вмешался, команды модели игнорируются.
- **Логировать входы и решения** — они отображаются оператору в мониторинге моделей и нужны для разбора.

????-??????: ??????? ?????????

Условие	Команда
Напряжение АКБ ниже 11 В	`stop`
Модель видит красный прямоугольник	`stop`
Модель видит белый прямоугольник	`forward`

Порядок проверки важен: безопасность (АКБ) → запрещающий сигнал (красный) → разрешающий (белый).

```
# policy.py — правила демо-прототипа
import cv2
import numpy as np

BATTERY_MIN_V = 11.0      # ниже — «стоп»
MIN_AREA = 0.02           # минимальная доля кадра, занимаемая прямоугольником

def find_rectangle(mask, frame_area):
    """True, если на бинарной маске есть крупный четырёхугольник."""
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    for c in contours:
        if cv2.contourArea(c) < MIN_AREA * frame_area:
            continue
        approx = cv2.approxPolyDP(c, 0.04 * cv2.arcLength(c, True), True)
        if len(approx) == 4 and cv2.isContourConvex(approx):
            return True
    return False

def see(frame):
    """Возвращает {'white': bool, 'red': bool} для BGR-кадра."""
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    area = frame.shape[0] * frame.shape[1]
    white = cv2.inRange(hsv, (0, 0, 200), (180, 40, 255))
    red = cv2.inRange(hsv, (0, 120, 80), (10, 255, 255)) | \
```

```
          cv2.inRange(hsv, (170, 120, 80), (180, 255, 255))
    kernel = np.ones((5, 5), np.uint8)
    white = cv2.morphologyEx(white, cv2.MORPH_OPEN, kernel)
    red = cv2.morphologyEx(red, cv2.MORPH_OPEN, kernel)
    return {"white": find_rectangle(white, area), "red": find_rectangle(red, area)}

def decide(frame, telemetry):
    """Правила демо-прототипа. Возвращает команду или None (ничего не менять)."""
    battery = telemetry.get("battery")
    if battery is not None and battery < BATTERY_MIN_V:
        return "stop"                       # 1. разряжена АКБ
    if frame is None:
        return None
    seen = see(frame)
    if seen["red"]:
        return "stop"                       # 2. красный прямоугольник
    if seen["white"]:
        return "forward"                    # 3. белый прямоугольник
    return None
```

Проверка на синтетических кадрах:

```
import numpy as np, cv2
from policy import decide

def frame_with(color=None):
    f = np.full((480, 640, 3), 60, np.uint8)
    if color:
        cv2.rectangle(f, (200, 150), (440, 330), color, -1)
    return f

assert decide(frame_with((255, 255, 255)), {"battery": 12.0}) == "forward"
assert decide(frame_with((0, 0, 255)),     {"battery": 12.0}) == "stop"
assert decide(frame_with(),                {"battery": 12.0}) is None
assert decide(frame_with((255, 255, 255)), {"battery": 10.5}) == "stop"
```

Пороги цвета в HSV зависят от освещения полигона и камеры — откалибруйте их по записанным кадрам.

??????? ? YOLO

Вместо цветовых порогов можно использовать детектор. Обучите YOLO на классах `white_sign` и `red_sign` (или используйте результаты `YoloModule` из конвейера — порт `yolo.detect`):

```
from ultralytics import YOLO

model = YOLO("signs.pt")          # веса, обученные на знаках полигона
CONF = 0.5

def see_yolo(frame):
    result = model(frame, verbose=False)[0]
    names = {result.names[int(b.cls)] for b in result.boxes if float(b.conf) >= CONF}
    return {"white": "white_sign" in names, "red": "red_sign" in names}
```

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

??????? 1. ?????? ?? ????? ??????

Самый простой способ запустить демо — выполнить модель прямо в основном цикле робота. Решение модели применяется без сетевой задержки, а в телеметрию добавляется, что «видит» модель — оператор увидит это в интерфейсе.

```
import cv2
from artex_robot import Robot
from policy import decide, see

robot = Robot({"type": "Robot", "name": "demo-01", "telem_can": True, "video_can": True})
robot.log.set_level("skip_TODO")
cam = cv2.VideoCapture(0)
robot["auto"] = False            # автопилот включается командой auto_run
robot["battery"] = 12.0

@robot.com_handler
def on_command(type_command, command, author=None):
    if command == "auto_run":
        robot["auto"] = True
```

```
    elif command == "auto_stop":
        robot["auto"] = False
        apply("stop")
    else:
        robot["auto"] = False    # ручная команда оператора отключает автопилот
        apply(command)

@robot.telemetry("battery")
def battery():
    return robot["battery"]      # здесь — чтение АЦП

@robot.telemetry("ai")
def ai_state(decision, seen):
    return {"auto": robot["auto"], "decision": decision, **seen}

@robot.video("rgb")
def send_frame(frame):
    return frame

@robot.mainloop
def loop(ctx):
    battery()
    ok, frame = cam.read()
    if not ok:
        return
    send_frame(frame)
    decision = decide(frame, {"battery": ctx["battery"]})
    ai_state(decision, see(frame))
    if ctx["auto"] and decision and decision != ctx.get("last_decision"):
        apply(decision)
        ctx["last_decision"] = decision

def apply(command):
    robot.log("INFO", f"Выполняю {command}")
    # здесь — управление моторами (см. «ПО для робота» → «Примеры роботов»)

robot.init_connector()
robot.start()
```

Команды `auto_run` / `auto_stop` вводятся в поле команд на странице управления роботом в веб-интерфейсе.

??????? 2. ?????? ? ?????????????? ????? ?????????

Модель упаковывается в Docker-образ и запускается в вычислительной среде (AI Runtime). Платформа подаёт на вход модели актуальный кадр и вектор телеметрии выбранного робота и передаёт её команды роботу через ХАБ. Рекомендуемая структура модели — класс с явными методами входа и выхода, независимый от транспорта:

```
class DemoModel:
    name = "demo-rules"
    version = "1.0"

    def __init__(self):
        self.last = None

    def step(self, frame, telemetry):
        """Вызывается платформой на каждое обновление данных. Возвращает команду или None."""
        cmd = decide(frame, telemetry)
        if cmd == self.last:
            return None          # не повторять ту же команду
        self.last = cmd
        return cmd
```

```
FROM python:3.11-slim
RUN pip install --no-cache-dir opencv-python-headless numpy
WORKDIR /model
COPY policy.py model.py ./
CMD ["python", "model.py"]
```

> Программный интерфейс подключения внешних моделей к ХАБу (методы `get` / `push` общей памяти, WebSocket/WebRTC-каналы) формируется на этапе прототипа «Альфа». Реализуйте логику модели как независимый класс, как в примере выше, — тогда перенос на итоговый транспорт сведётся к небольшому адаптеру.

??????? 3. ?????? ? ?????????

Для типовых задач (детекция YOLO) модель подключается как модуль в визуальном редакторе без программирования — см. Конвейеры модулей.

???-???? ????? ???????? ?????? ?? ???????

- ☐ Модель проверена на записанных данных полигона (кадры + телеметрия).
- ☐ Время решения укладывается в бюджет (замерьте на целевом оборудовании).
- ☐ При отсутствии кадра или телеметрии модель возвращает безопасное решение (`None` или `stop`).
- ☐ Команды модели соответствуют словарю команд робота.
- ☐ Проверено, что вмешательство оператора перекрывает модель.
- ☐ Логирование входов и решений включено.

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

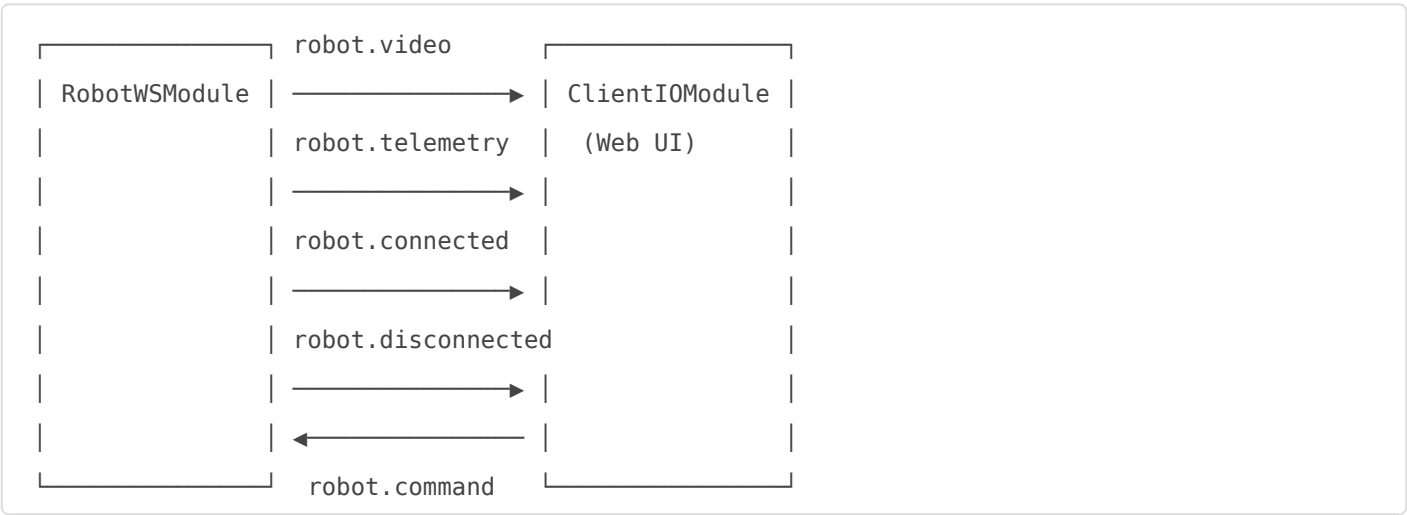
Логика обработки данных на сервере собирается в визуальном редакторе модулей: модули перетаскиваются на рабочее поле, их порты соединяются линиями. Описание самих модулей и их портов — на странице Веб-интерфейс управления роботами; здесь — готовые схемы соединения для типовых задач.

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

- Данные идут от **выхода** одного модуля ко **входу** другого; соединяются порты одного типа (`robot.video` → `robot.video`).
- Один выход можно соединить с несколькими входами — например, видео одновременно идёт в интерфейс и в YOLO.
- Изменения конвейера применяются без перезапуска сервера (hot reload).
- Для отладки подключите **ConsoleLogModule** к интересующему порту — все данные будут выведены в консоль сервера.

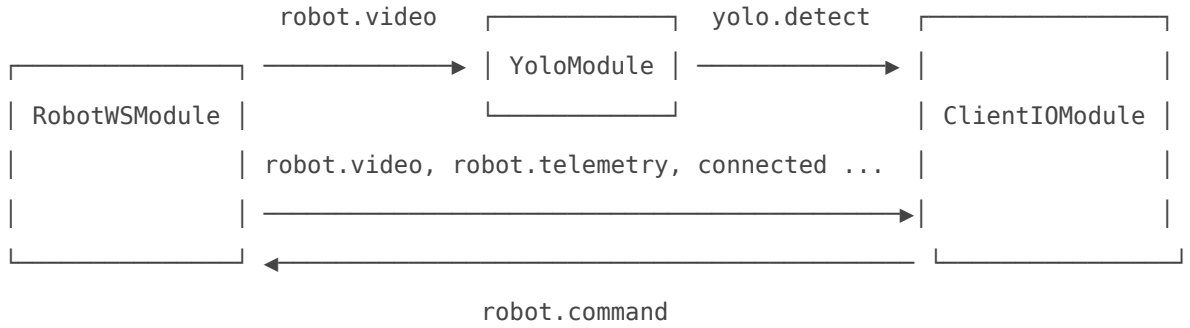
????? 1. ?????? ??????????? (???????????)

Оператор видит видео и телеметрию робота и управляет им.



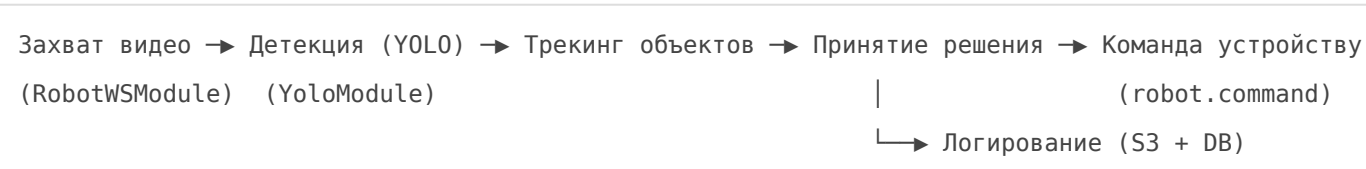
????? 2. ???????? ???????? ?????? ?????

Оператор управляет вручную, а YOLO подсвечивает найденные объекты.



????? 3. ?????????? ?????????? ?? ?????

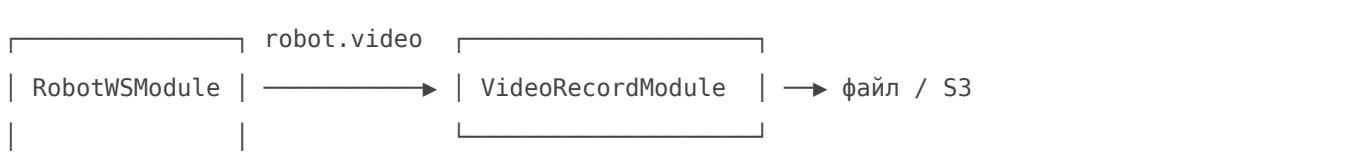
Конвейер полного автономного контура: захват → детекция → трекинг → решение → команда, с логированием.



Запуск и остановка автоматического управления — командами `auto_run` / `auto_stop` из поля ввода команд на странице робота. Оператор в любой момент может нажать «Стоп».

????? 4. ?????? ????????

Для сбора данных к источнику видео и телеметрии подключается модуль записи (VideoRecordModule). Запись включается командой `video_start_record` и останавливается командой `video_stop_record`.



```
│                │ robot.video, robot.telemetry ──────► ClientIOModule
└────────────────┘
```

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

Проблема	Проверка
В интерфейсе нет видео	Соединён ли `robot.video` источника с ClientIOModule? Подключите ConsoleLogModule к `robot.video`
YOLO не выдаёт результатов	Идёт ли видео на вход YoloModule; есть ли объекты известных модели классов
Команды не доходят до робота	Соединён ли `robot.command` ClientIOModule с RobotWSModule; получил ли робот `Ready`
Высокая задержка	Не подключено ли слишком много тяжёлых модулей к одному потоку; загрузка GPU в мониторинге

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

Сбор датасетов, Imitation Learning, RL, полигоны

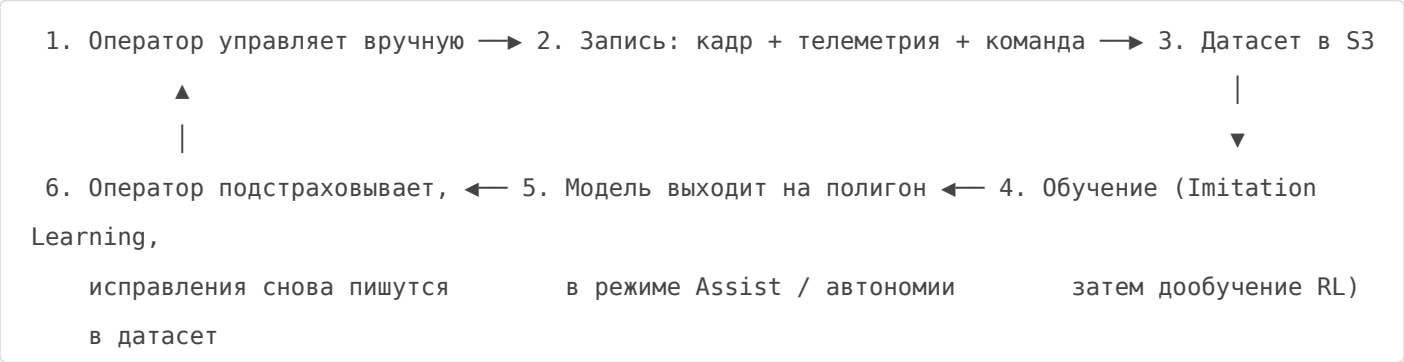
Данные и обучение

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

???????

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».