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

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.

Revision #3
Created 2026-09-23 09:32:28 UTC by Маслов Алексей
Updated 2026-09-23 09:35:10 UTC by Маслов Алексей