

# Летний интенсив 2023

## Компьютерное зрение ROS2

**Skoltech**

Skolkovo Institute of Science and Technology



School of robotics

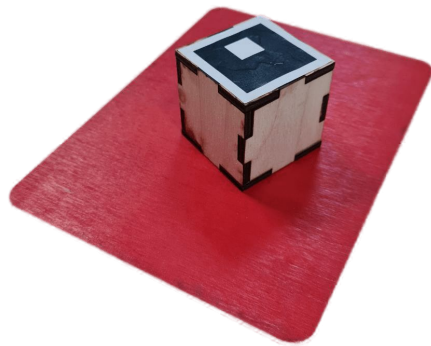
Преподаватели курса:

- Бурмистров Степан
- Федосеев Алексей

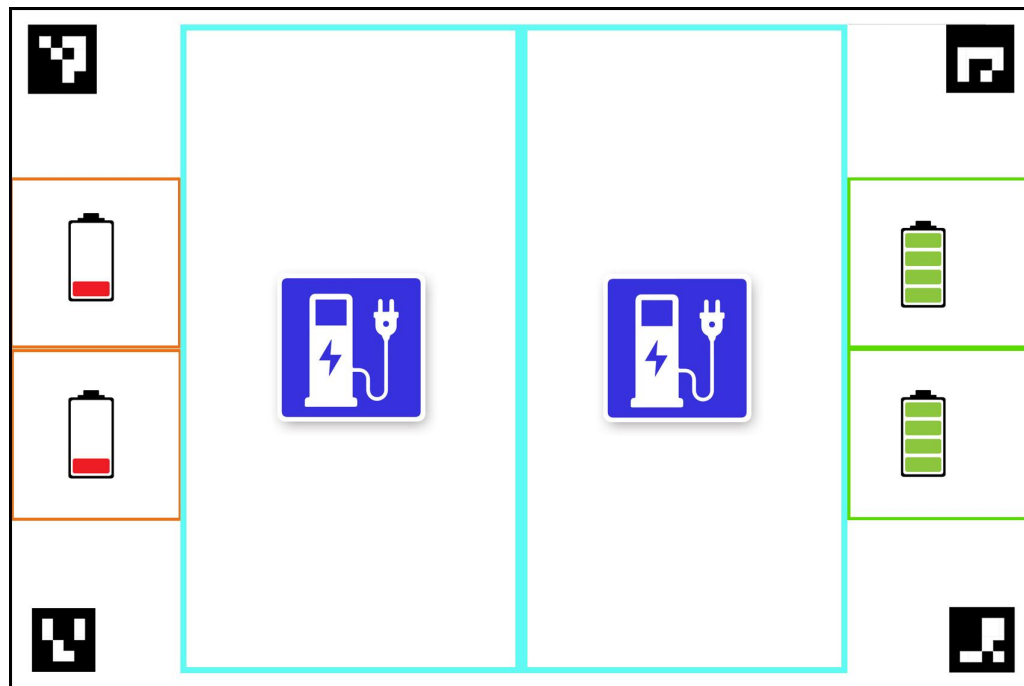
Руководитель  
лаборатории:

- Дмитрий Тетерюков

## Автоматизированная станция по замене АКБ на роботах



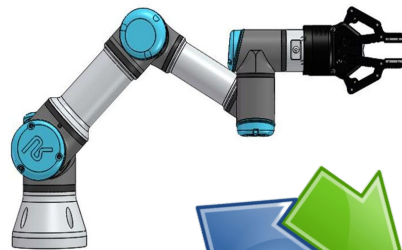
Модель робота и  
АКБ



# Выполненные задачи:



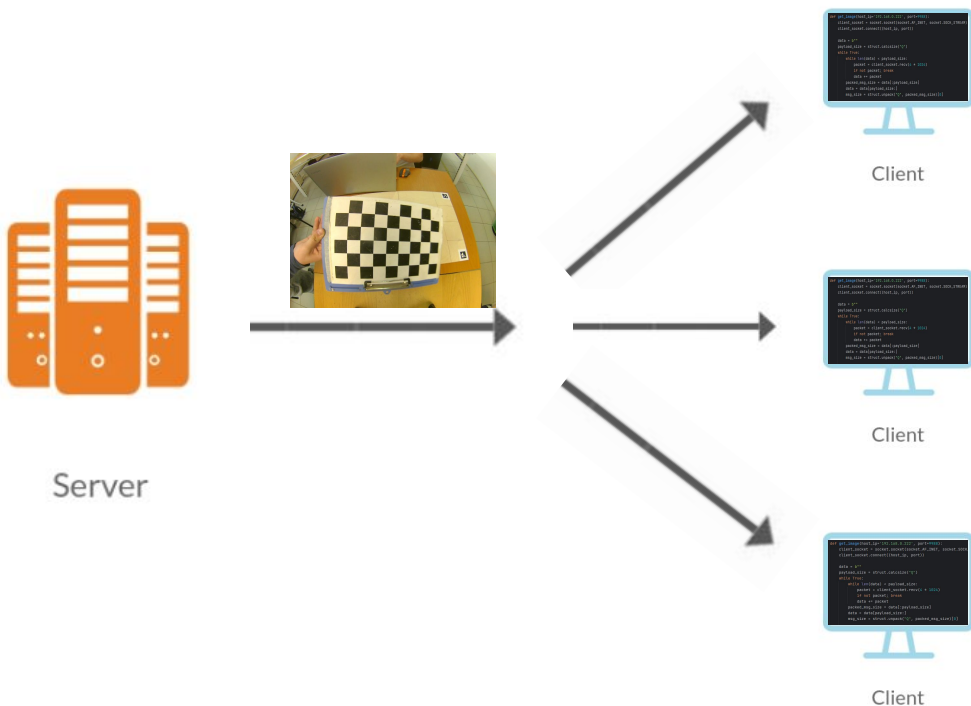
- Создание сервера для общего доступа к камере
- Поиск рабочего поля и объектов на нем
- Совмещение систем координат
- Создание маршрутов движения



- Создание системы удаленного управления
- Установка и внедрение ROS2 Humble
- Настройка VPN-сервера для доступа
- Разработка WEB-сервера для контроля манипулятора



# Получение изображения



как сделать фото с помощью веб  
камеры? у меня ноутбук и windows  
7.. ?

ответы

Как написать код на Python,  
который будет выводить  
изображения с камеры телефона  
(андроид) на монитор?

ответы

AR

Как запитонить изображение с камеры?



Хаха, конечно, давай я расскажу тебе, как запитонить изображение с камеры! Во-первых, ты возьми змею-питона, привяжи её хвост к камере, чтобы она могла таскать её с собой. Затем научи змею-питона считать каждый пиксель изображения с помощью своих чешуйчатых лапок. Далее, позволь змее-питону проглотить каждый пиксель и превратить его в символы на своём теле. Таким образом, ты "запитоняешь" изображение с камеры, превращая его в чешуйчатую змею-питона! Теперь ты можешь гордиться своим новым способом обработки изображений в стиле Python!

# Калибровка камеры

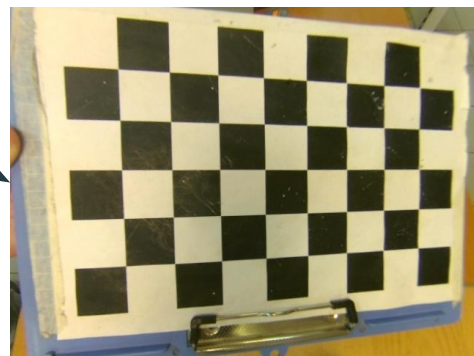
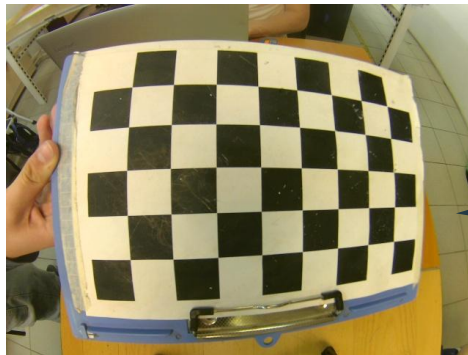
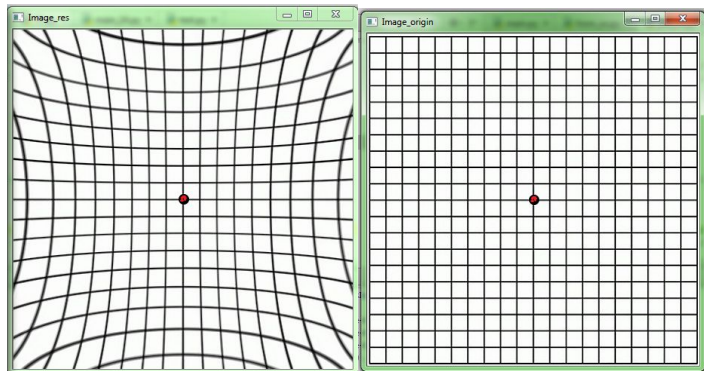
- Создание датасета с изображениями шахматных досок
- Получение параметров калибровки и сохранение их в файл
- Корректирование изображения камеры



Параметры калибровки в файле:

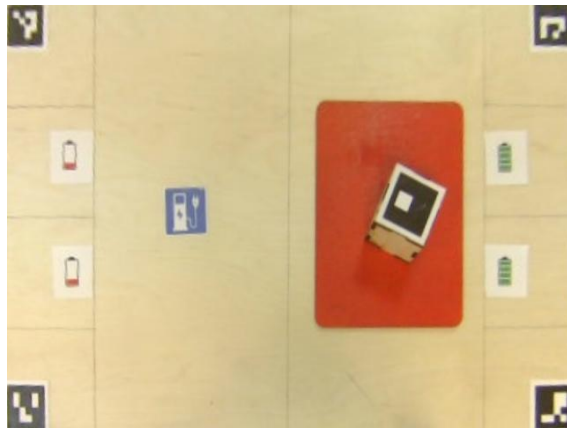
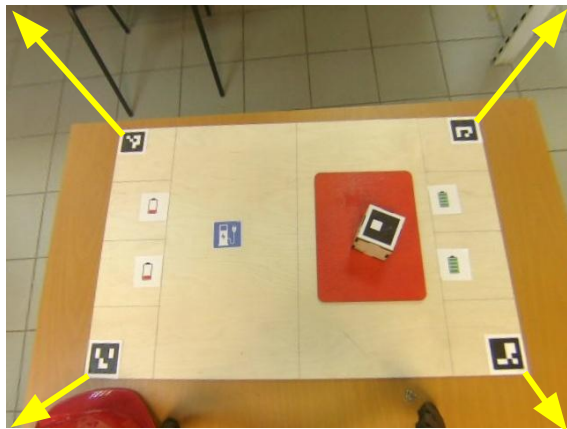
```
np.array([[363.98281793, 0.0, 321.679658058], [0.0, 362.9350007146, 287.1295250852776], [0.0, 0.0, 1.0]])  
np.array([[0.0104022392], [-0.1082444188826], [0.1732694353821679], [-0.09974663798380133]])
```

# Корректирование изображения



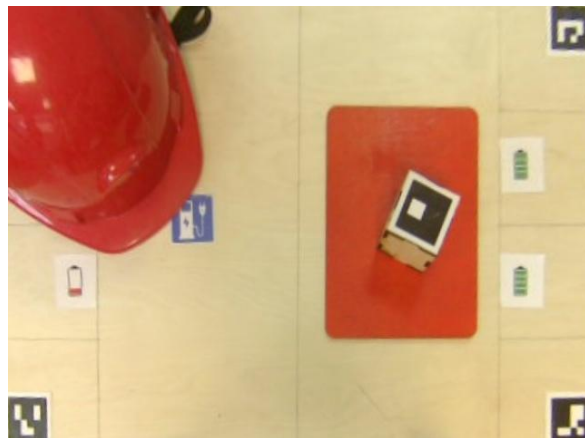
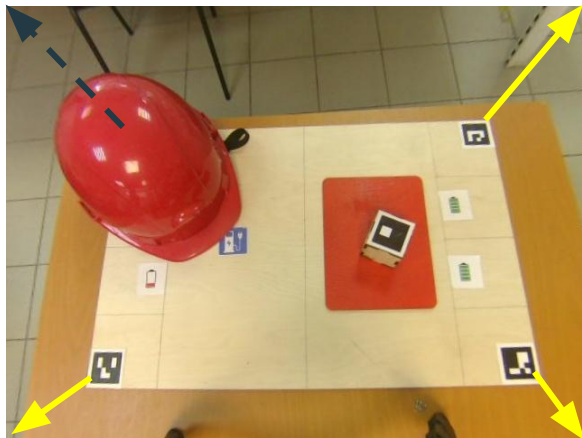
```
DIM = img.shape[:2][::-1]
map1, map2 = cv2.fisheye.initUndistortRectifyMap(K, D, np.eye(3), K, DIM, cv2.CV_16SC2)
undistorted_img = cv2.remap(img, map1, map2, interpolation=cv2.INTER_LINEAR, borderMode=cv2.BORDER_CONSTANT)
```

## Создание псевдо-вида сверху





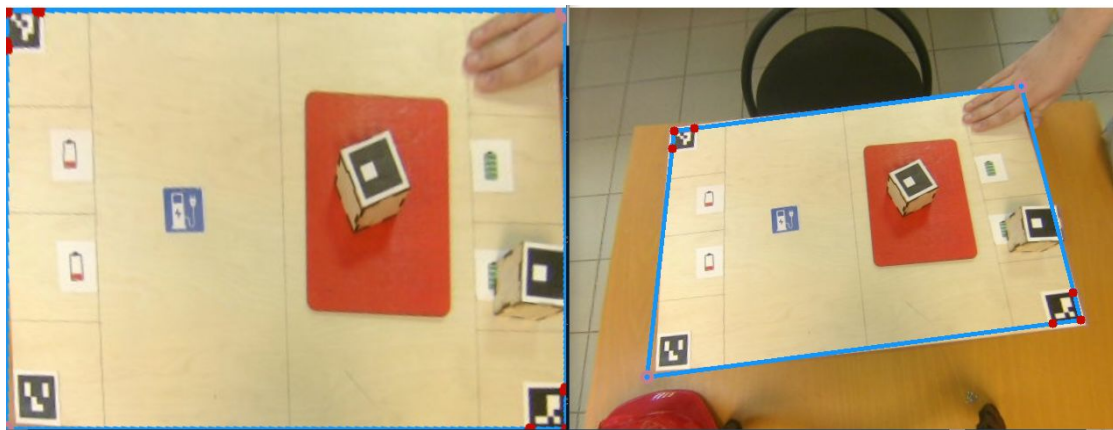
# Метод сохранения данных





# Линейные функции

$$y=kx+b$$



$$k = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y_2 - k * x_1$$

$$x = \frac{b_2 - b_1}{k_2 - k_1}$$

$$y = k_2 - x * b_1$$

# Нахождения ArUco маркера на поле

```
res2 = cv2.aruco.detectMarkers(gray2, dictionary)
```

Использование списка  
cv2.aruco.detectedMarkers()

```
for i in range(len(np.where(res2[1] == num)[0])):
```

```
    index = np.where(res2[1] == num)[0][i]
```

```
    pt0 = res2[0][index][0][0].astype(np.int16)
```

```
    pt1 = res2[0][index][0][1].astype(np.int16)
```

```
    pt2 = res2[0][index][0][2].astype(np.int16)
```

```
    pt3 = res2[0][index][0][3].astype(np.int16)
```

```
    center = [(pt0[0] + pt2[0])//2, (pt0[1] + pt2[1])//2]
```

```
    tmp_angle = 0
```

```
    kat_0 = pt1[0] - pt0[0]
```

```
    kat_1 = pt1[1] - pt0[1]
```

```
    tg = kat_1 / kat_0
```

```
    tmp_angle = math.atan(tg)
```

```
    tmp_angle = math.degrees(tmp_angle)
```

```
    tmp_angle = math.fabs(tmp_angle)
```

```
    tmp_angle = int(tmp_angle)
```

```
    pt0 = list(pt0)
```

```
    pt1 = list(pt1)
```

```
    pt2 = list(pt2)
```

```
    pt3 = list(pt3)
```

```
    output.append([center, tmp_angle, pt0, pt1, pt2, pt3])
```

Получаем угловые точки  
маркера

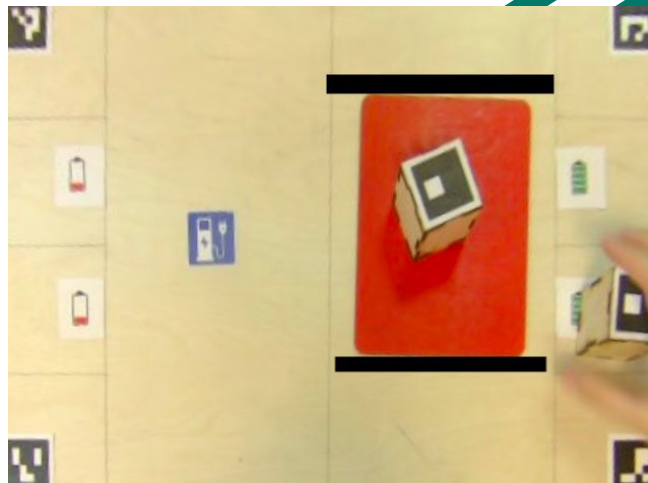
Находим центр

Находим угол поворота

Преобразуем объекта numpy в  
список

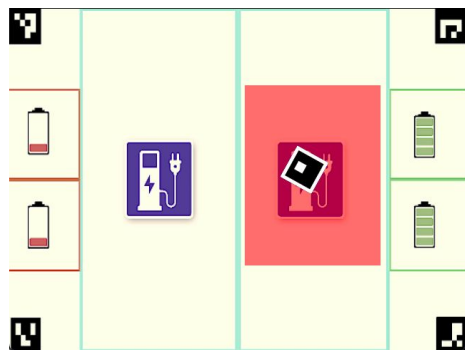
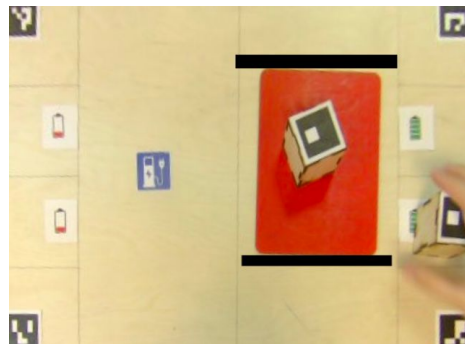
# Находим верхнюю и нижнюю грань робота

```
def find_robot(robot_zone_img):  
    h_min = (0, 108, 170)  
    h_max = (255, 255, 255)  
    robot_zone_img = cv2.cvtColor(robot_zone_img,  
        cv2.COLOR_RGB2HSV)  
    img_bin_r = cv2.inRange(robot_zone_img, h_min, h_max)  
  
    kernel = np.ones((5, 5), 'uint8')  
    thresh = cv2.erode(img_bin_r, kernel, iterations=4)  
    img_bin_r = cv2.dilate(thresh, kernel, iterations=3)  
  
    summa = np.sum(img_bin_r, axis=1)  
    dafk = np.where(summa > 10000)  
  
    if len(dafk[0]) > 1:  
        return dafk[0][0], dafk[0][len(dafk[0]) - 1]  
    else:  
        return None, None
```



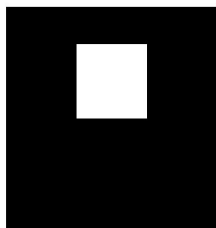
- Находим диапазоны цвета робота
- Убираем шумы операциями dilate, erode
- Суммируем абсциссу
- Находим значение списка > 10000
- Возвращаем крайние элементы списка

# Визуализируем поле

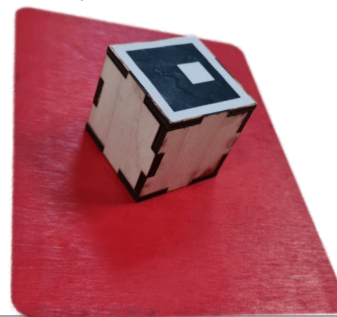


- Выводим картинку поля
- Рисуем робота
- Рисуем аруко маркеры с углом поворота
- Рисуем зеленые квадраты, если есть маркер
- Рисуем красные квадраты, если нет маркера

аруко маркер



робот



# Управление роботом

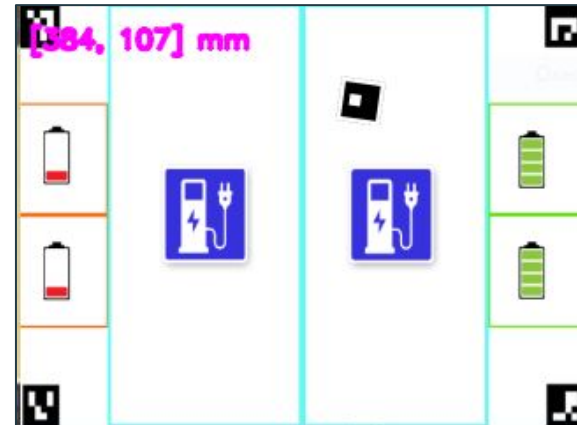
```
def move_robot_l(x, y, z, angle):  
    rx, ry, rz = angle_gripper(angle)  
    left_robot.move1([x, y, z, rx, ry, rz], 0.1, 0.2, wait=True)  
  
def move_robot_r(x, y, z, angle):  
    rx, ry, rz = angle_gripper(90+angle-0.01)  
    right_robot.move1([x, y, z, rx, ry, rz], 0.1, 0.2,  
wait=True)
```



# Нахождение положения батарей робота в координатах поля

Важная задача для управления роботом - это нахождение положения батарей в координатах поля, иначе говоря, перевод пикселей в мм. Для решения данной задачи применяется несколько различных методов, данные которых комплексированы. Методы для нахождения:

- Нахождение с использованием линейной функции
- Нахождение координат боковой грани



# Нахождение положения батареи с использованием линейной функции

При исследовании искажений батарей было выявлено, что смещение изображения верхней грани относительно изображения нижней грани прямо пропорционально расстоянию от центра поля для оси абсцисс и расстоянию от нижнего края поля для оси ординат, т. е. перевод пикселей в миллиметры можно осуществить с помощью математической линейной функции.

Были вычислены  $k$  - коэффициенты для:  
OX: 1.333  
OY: 1.560

Таким образом можно найти положение батареи в координатах поля с помощью формулы:

$$x = \frac{y}{k}$$

где:  $x$  - координаты в мм,  
 $y$  - координаты в пкс,  
 $k$  - коэф.

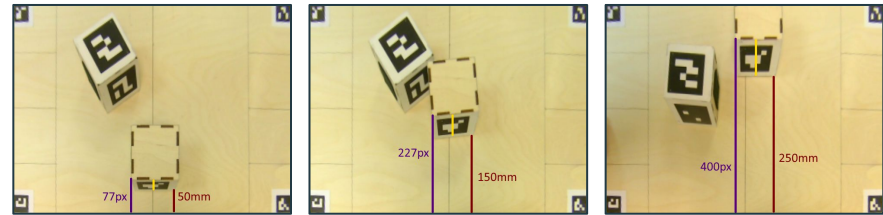
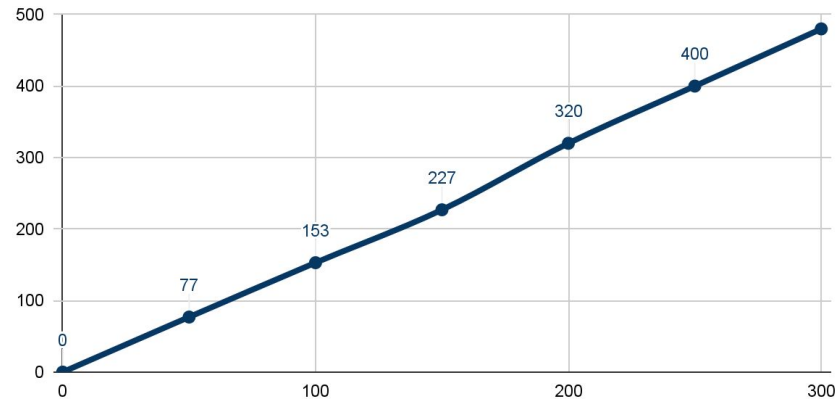


График значений по OY, полученных при исследовании изображения





# Нахождение координат батареи через поиск нижней стороны боковой границы

Для нахождения координат батареи нужно найти координаты нижней стороны боковой грани батареи (в пкс.) и перевести их в мм;

при этом опытным путём было вычислено, что с текущим разрешением и углом обзора камеры:  $1 \text{ пкс.} = 0.75 \text{ мм.}$



# Комплексирование данных для получения положения батарей робота в координатах поля

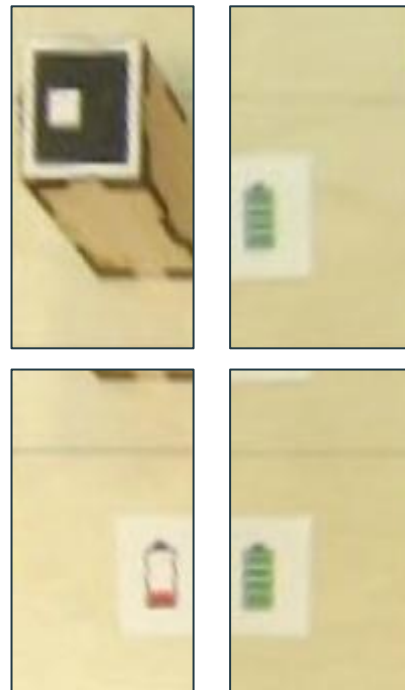
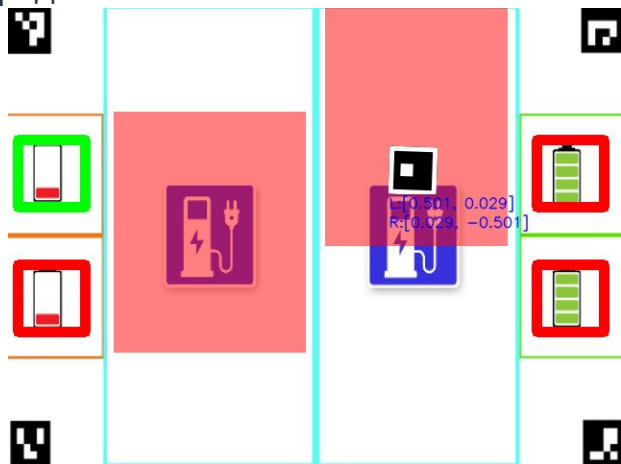
Для комплексирования данных, полученных различными методами, используется среднее арифметическое соответственных координат, т. е. при комплексировании двух массивов данным образом, на выходе будет получен массив, где каждый  $n$  член будет равняться среднему арифметическому всех  $n$  членов входных массивов.

```
>>> list_average ([1, 2], [3, 4])  
[2, 3]
```

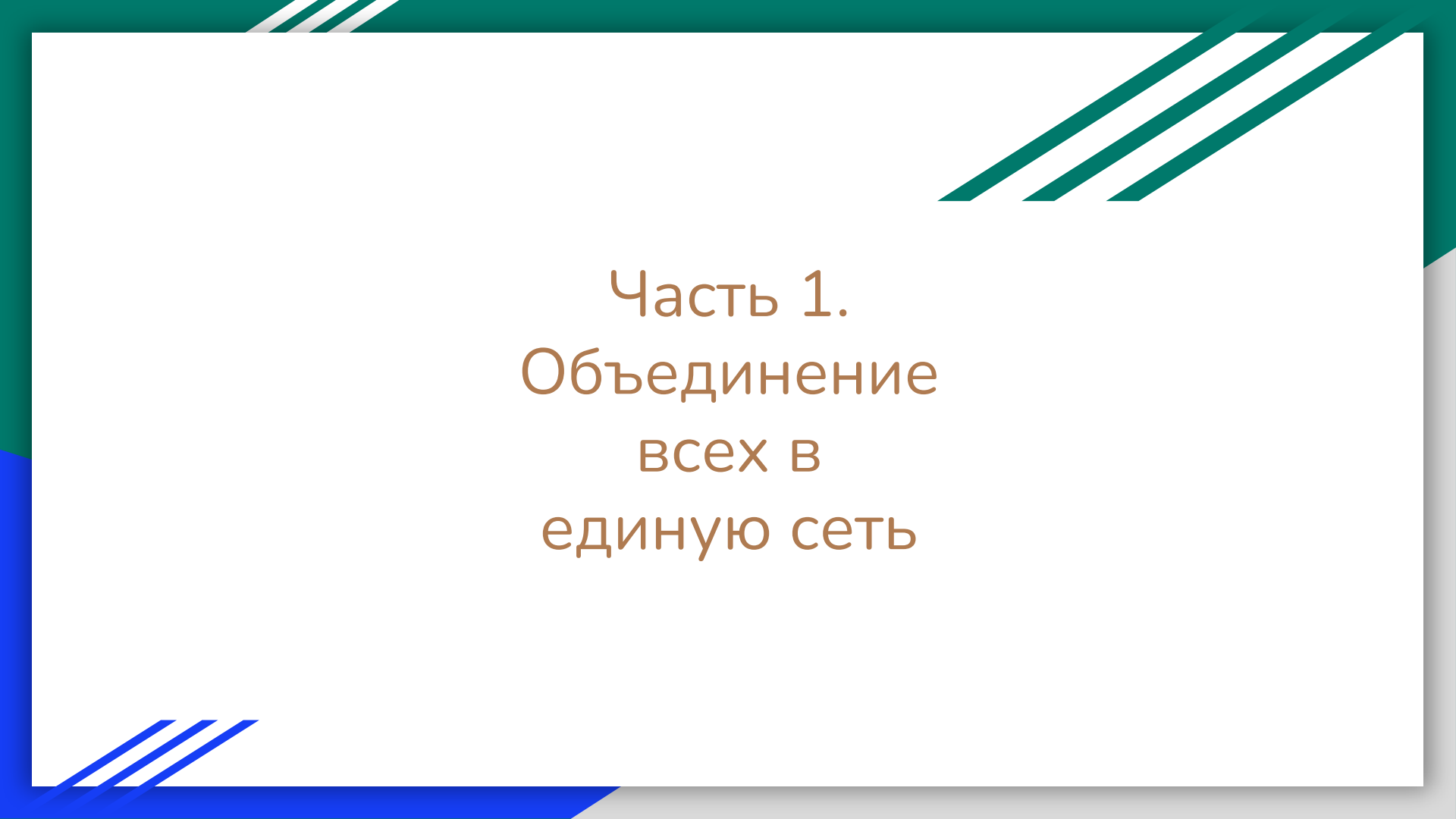
После комплексирования всех данных, полученных ранее описанными методами, *на выходе мы получаем положение батареи робота в координатах поля.*

# Определение наличия батарей в зоне зарядки-разрядки

Для управления роботом производится проверка наличия батарей в поле зарядки-разрядки.

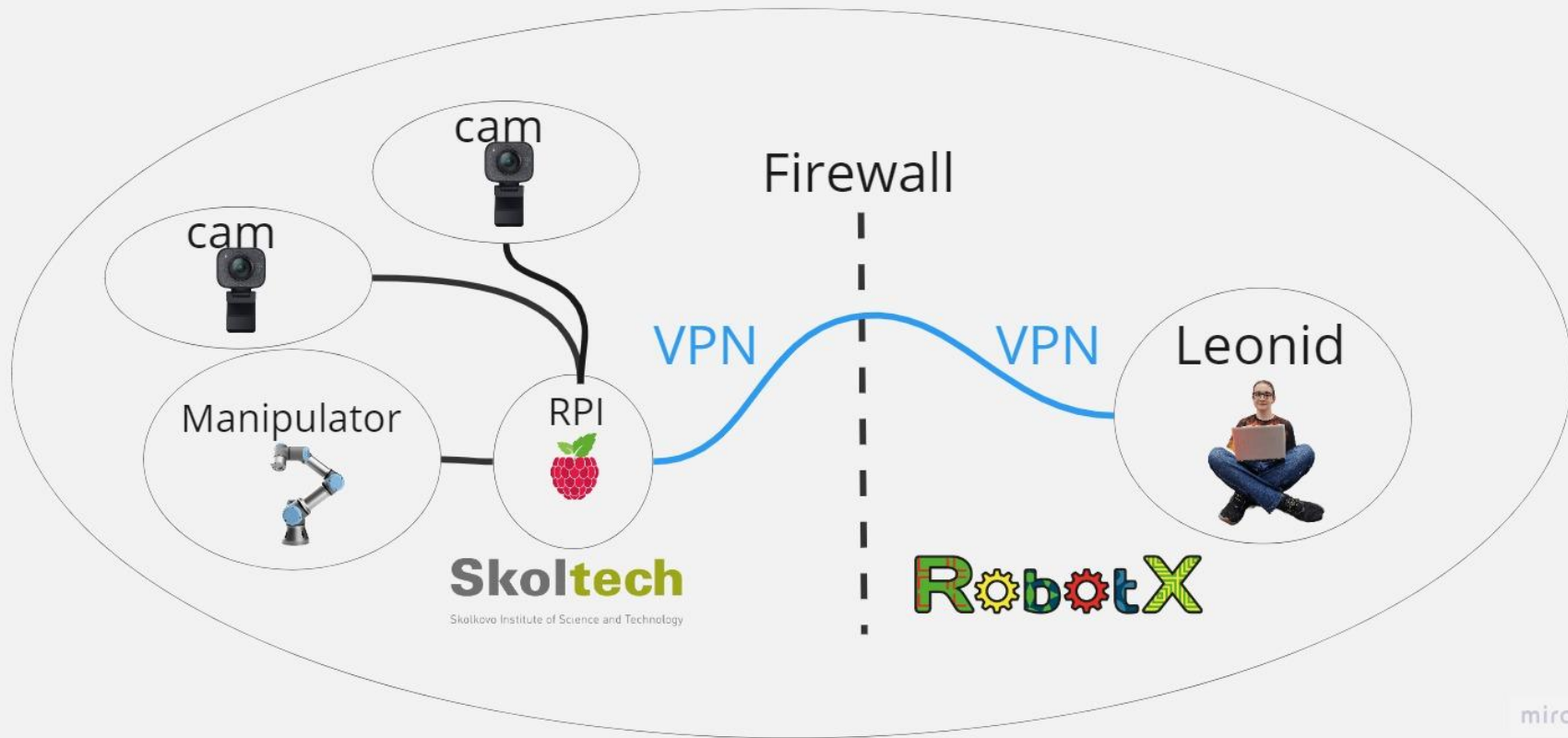


# Удаленный доступ к манипулятору с ROS2



# Часть 1. Объединение всех в единую сеть

# Схема удаленного доступа



## Используемые технологии





Инструкция:

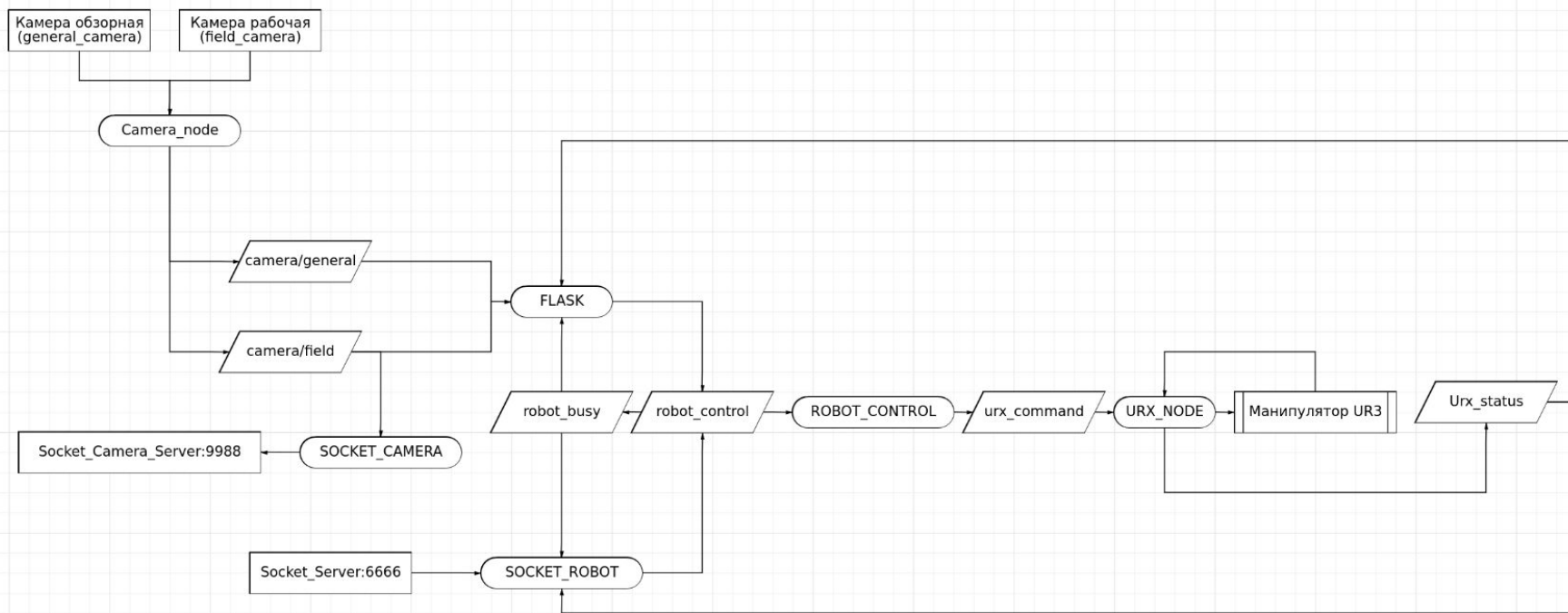


<https://dzen.ru/a/ZJ1CO7Y17y8tswpD>

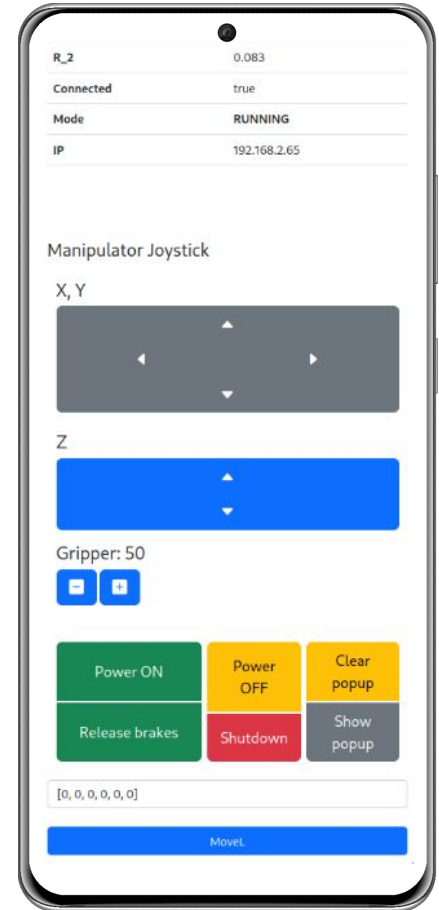
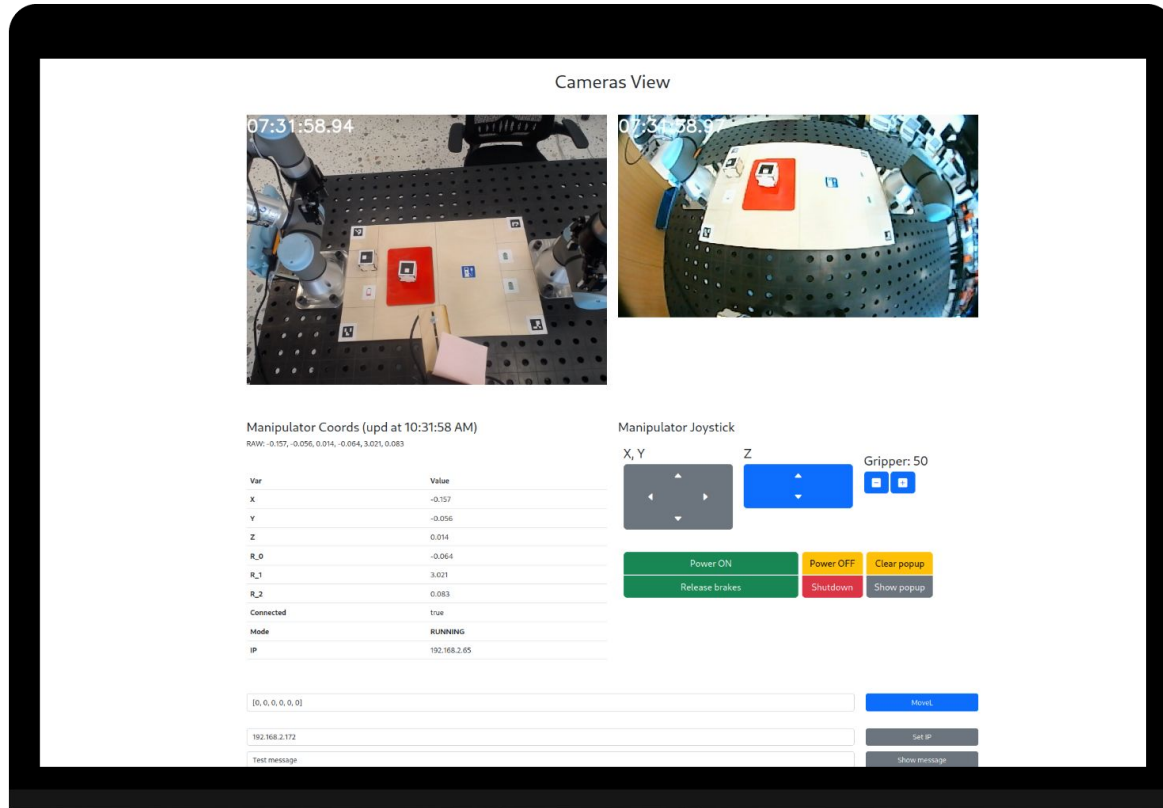


## Часть 2. ROS2, Web интерфейс

# Общая архитектура



# WEB сервер на Flask



# TCP socket interfaces

## Socket robot - TCP on 6666

Input: command manipulator  
getl, movel, gripper\_get,  
gripper\_set

Output: gripper ||  
robot position

JSON:

```
{  
  "type": string,  
  "pose": list || int  
}
```

JSON:

```
{  
  "pose": list || int  
}
```

## Socket camera - TCP on 9988

Output: image

Pickled opencv image from /camera\_field  
3 FPS per client



# Robot control

Control command

Robot control check

Robot busy check

```
{  
  "type": "move1",  
  "pose": [-0.157, -0.056,  
0.014, -0.064, 3.021, 0.083]  
}
```



Safe command

Robot free

URX node

```
{  
  "type": "move1",  
  "pose": [1, -10, 50, 1,  
2, 2]  
}
```



Unsafe command

Ignore

# UR3 Dashboard Server

`sudo apt install netcat putty` - установка необходимых пакетов на устройство для управления (команды *nc* и *plink*)

Connect to manipulator ssh  
*root:easybot*



Send command with netcat in ssh

Шаблон для выполнения команды для контроля робота:

```
echo y | plink root@ROBOT_IP -pw easybot "{ echo "COMMAND"; echo "quit"; } | nc 127.0.0.1:29999"
```

## Основные команды

*robotmode* - статус робота

*power on* - включение питания

*power off* - выключение питания

*shutdown* - полное отключение питания

*brake release* - отключение тормозов

*close popup* - закрыть уведомление

*popup TEXT* - показать уведомление `TEXT`



# ROS params

## Camera node

```
/camera_node:
  ros__parameters:
    general_camera_path:
      "/dev/v4l/by-path/platform-fd50
0000.pcie-pci-0000:01:00.0-usb-0:1.1:1.
0-video-index0"
    field_camera_path:
      "/dev/v4l/by-path/platform-fd50
0000.pcie-pci-0000:01:00.0-usb-0:1.3:1.
0-video-index0"
    publish_timer: 0.03 # 10 fps
    frame_width: 640
    frame_height: 480
```

## Flask node

```
/flask_node:
  ros__parameters:
    port: 8080
    host: "0.0.0.0"
    joystick_offset: 0.01 # m
```

## URX node

```
/urx_node:
  ros__parameters:
    ip: "192.168.2.65"
    popup_message: "Манипулятор
управляется дистанционно"
    status_publish_rate: 0.5 #
s
    gripper_start_pose: 0 # 0 -
255
    gripper_step: 10 # 0 - 255
```

## Robot control node

```
/robot_control:
  ros__parameters:
    gripper:
      max: 100
      min: 0
    move1: # m
      x_min: -0.22
      x_max: 0.22

      y_max: -0.150
      y_min: -0.5

      z_min: 0.17
      z_max: 0.35

    velocity: 0.15
    acceleration: 0.15
```

# Спасибо за внимание!



<https://github.com/robotx-school/Remote-Manipulator>

<https://github.com/robotx-school/CV-June-2023>



<https://dzen.ru/a/ZJ1CO7Y17y8tswpD>

**Skoltech**

Skolkovo Institute of Science and Technology



School of robotics



alexeyfas



stepan\_burmistrov