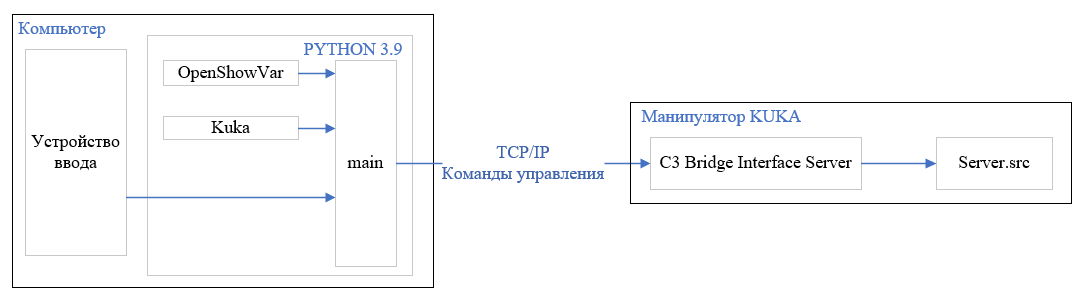
# Дистанционное управление роботом манипулятором Kuka

# Установка программного обеспечения на манипулятор

## Архитектура программного комплекса

Общая архитектура программного комплекса для дистанционного управления представлена на рисунке 1-1.



**Рис.1-1: Архитектура программного комплекса**

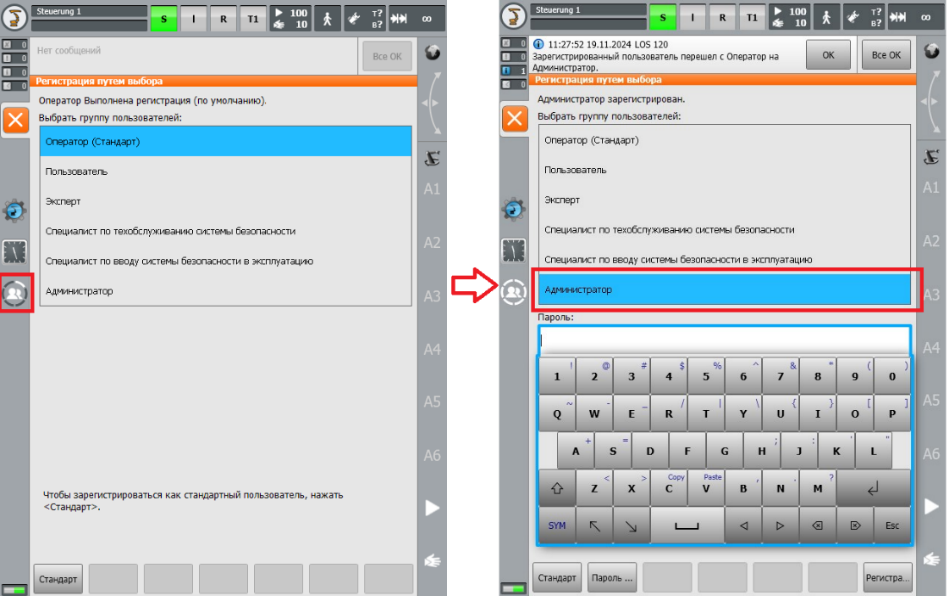
В качестве устройства ввода может выступать:

* микрофон – при организации диалоговой системы управления;
* клавиатура – в случае удаленного ручного управления роботом манипулятором;
* камера – при реализации систем управления на основе технического зрения.

С помощью модели протоколов TCP/IP на промышленный манипулятор KUKA передаются команды управления через C3 Bridge Interface Server, который позволяет удаленным клиентам выполнять запросы к подсистеме KUKA Cross 3 и возвращать ответы. Контроллеры Kuka оснащены двумя операционными системами: ОС VxWorks для управления манипулятором в реальном времени и ОС Windows для визуализации. Эти два уровня взаимодействуют друг с другом. Например, для отображения текущих координат объекта «POS\_ACT» на KUKA smartPAD требуется запросить у ОС реального времени текущее значение $POS\_ACT. Эта связь осуществляется через перекрестный интерфейс KUKA Cross 3. Server.src – программа, которая принимает команды управления от компьютера и передает их на исполнительные механизмы манипулятора.

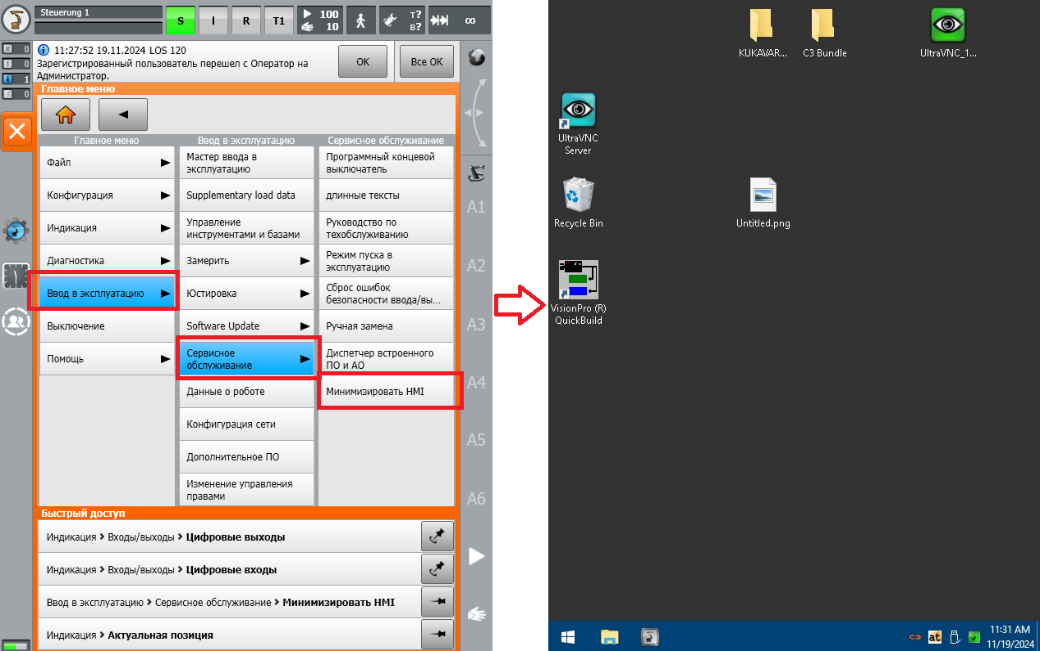
## Добавление 7000 порта на контроллер манипулятора

Для начала необходимо на контроллере манипулятора прописать порт в файл **KLIConfig.xml**. Для этого необходимо авторизоваться в системе манипулятора в качестве **Администратора**. Последовательность процедуры авторизации показана на рис.1-2.



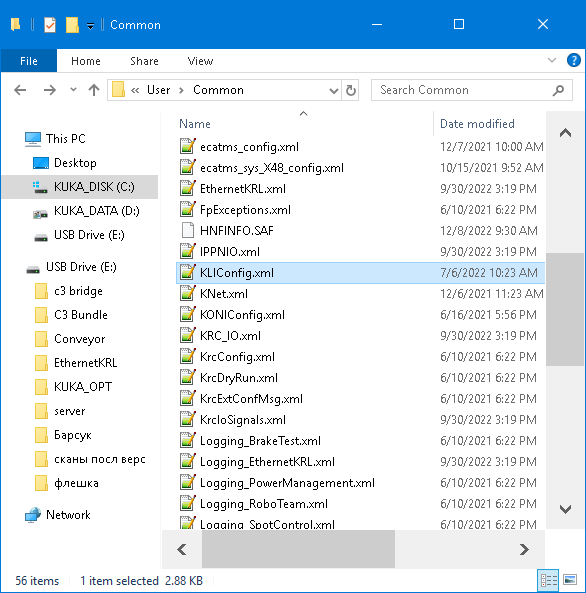
**Рис.1-2: Процедура перехода в Администратора**

После ввода пароля **администратора** необходимо перейти в **ОС Windows**. Последовательность перехода приведена на рис.1-3.



**Рис.1-3: Процедура перехода в ОС Windows**

В ОС Windows файл **KLIConfig.xml** расположен в *C:\KRC\ROBOTER\Config\User\Common* (Рис.1-4).



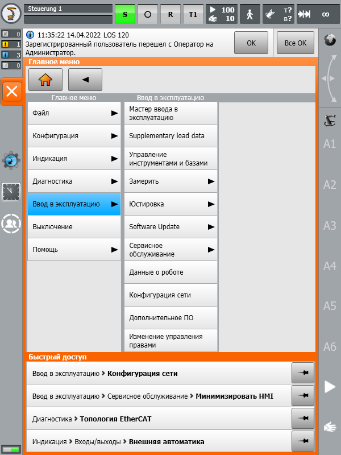
**Рис.1-4: Директория с файлом KLIConfig.xml**

С помощью текстового редактора Notepad++ необходимо добавить **7000-ый** порт, как показано на рис.1-5.

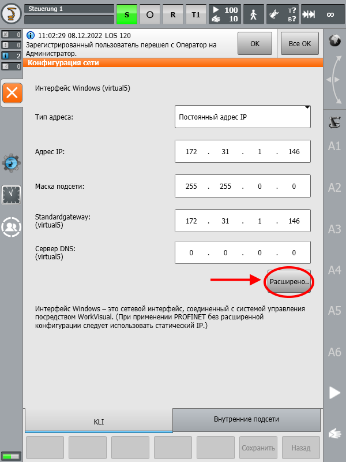


**Рис.1-5: Добавление 7000-го порта**

После сохранения изменений в файле **KLIConfig.xml** необходимо перейти в раздел меню «**Ввод в эксплуатацию**» -> «**Конфигурация сети**» (рис. 1-6) и открыть расширенные настройки сети (рис. 1-7).

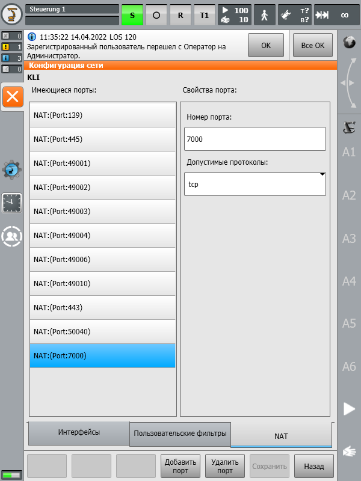


**Рис. 1-6: Главное меню KUKA SmartPAD**



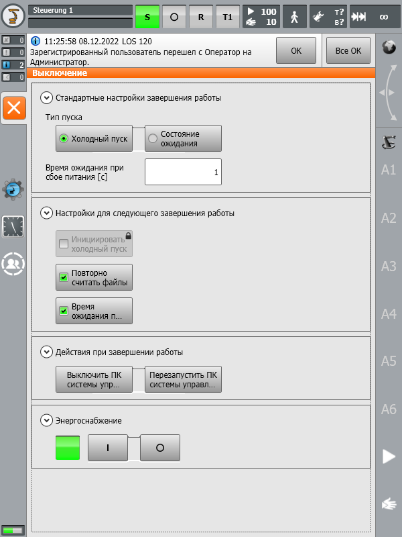
**Рис. 1-7: Настройка конфигурации сети**

В нижнем меню расширенных настроек сети необходимо перейти на вкладку **NAT** и по кнопке **«Добавить порт»** ввести номер порта **7000** и протокол tcp (рис. 1-8)

****

**Рис. 1-8: Добавление 7000-го порта**

Для сохранения проведенных изменений необходимо выполнить перезагрузку контроллера манипулятора. Для этого в главном меню необходимо нажать кнопку «**Выключение**». Если вы находитесь в режиме администратора появится опция выбора типа пуска «**Холодный пуск**». В настройках для следующего завершения работы необходимо поставить галочку «**Повторно считать файлы**» и в завершении выбрать команду «**Перезапустить ПК системы управления**» (Риc. 1-9).

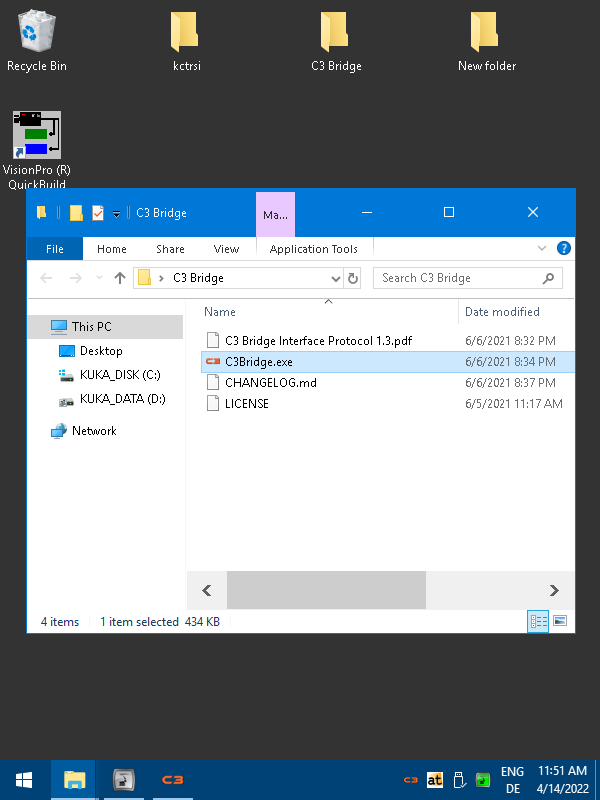


**Рис. 1-9: Окно выключения робота**

## Установка C3 Bridge Interface Server на контроллер манипулятора

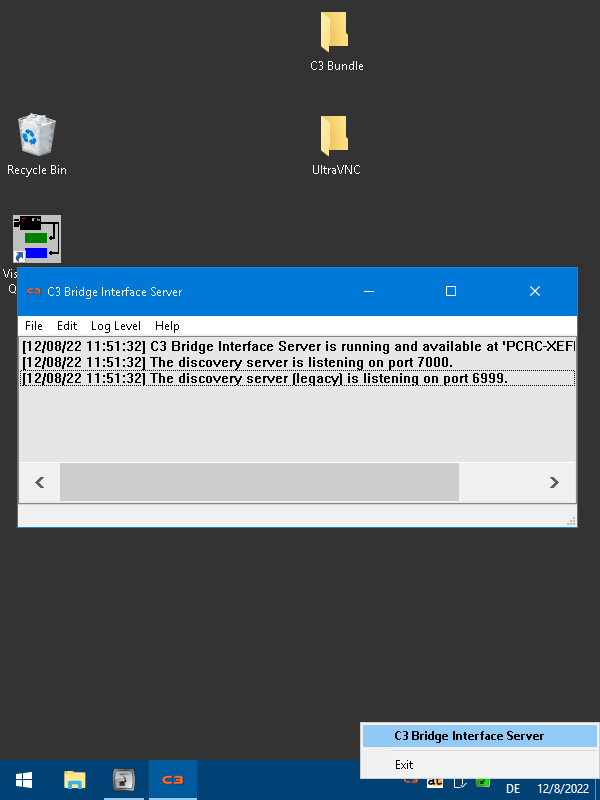
C3 Bridge Interface Server — это сетевое приложение, позволяющее удаленным клиентам выполнять запросы к подсистеме KUKA Cross 3 и возвращать ответы. Более подробно протокол взаимодействия описан по ссылке: <http://c3.ulsu.tech/protocol/latest/>.

Программа **C3 Bridge Interface Server**, поставляется вместе с ПО и ее необходимо загрузить на контроллер манипулятора. Для этого подключите флеш-карту с программой к контроллеру манипулятора и вновь перейдите в ОС Windows (рис.1-3). Папку с ПО необходимо переместить на рабочий стол и запустить исполнительный файл C3Bridge.exe (Рис. 1-10).



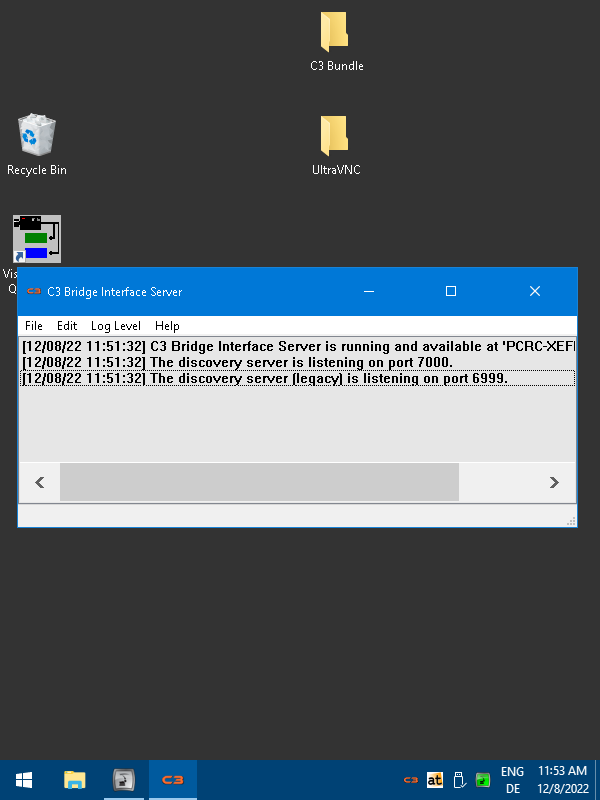
**Рис. 1-10: Запуск C3 Bridge Interface Server**

Для проверки открываем окно C3 Bridge Interface Server нажимая правой кнопкой мыши на иконку C3 в панели задач (Рис. 1-11).



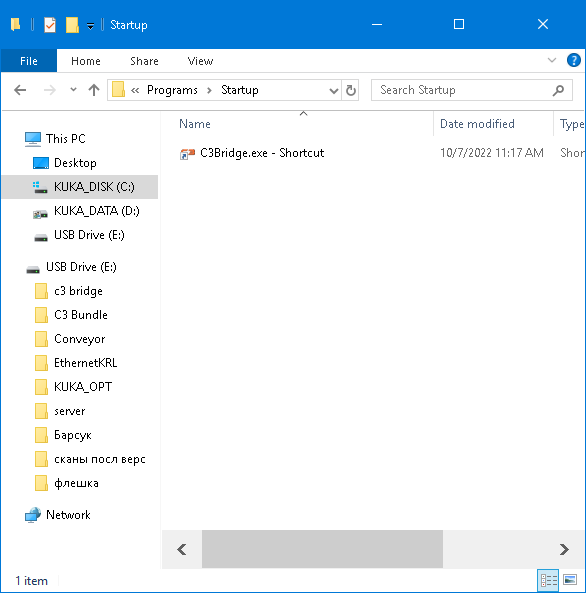
**Рис. 1-11: Вызов графического интерфейса C3 Bridge Interface Server**

После выполнения данной операции появится диалоговое окно C3 Bridge Interface Server (Рис. 1-12).



**Рис. 1-12: Графический интерфейс C3 Bridge Interface Server**

Чтобы не запускать C3 Bridge Interface Server каждый раз после включения робота, его необходимо добавить в автозапуск. Для этого необходимо создать ярлык C3Bridge.exe и разместить его в папке **Startup** (Рис.1-13). Директория расположена в *C:\Users\KukaUser\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup* (необходимо в настройках проводника включить отображение скрытых файлом и папок).

****

**Рис. 1-13: Директория Startup и ярлык C3Bridge**

## Редактирование файла конфигурации на контроллере манипулятора

На контроллере манипулятора KUKA системные переменные, которые имеют в начале знак «$», запрещены для редактирования в случае управления с внешнего устройства. Полный перечень системных переменных приведен по ссылке: <https://openkuka.github.io/krl/reference/system.types/>.

Для возможности использования системных переменных, в случае управления с внешнего устройства, их необходимо переопределить в программе, написанной на языке KRL, но предварительно их необходимо объявить. Для этого они должны быть прописаны в файле **Config.dat** на манипуляторе, расположенном в OC Windows в папке C:\KRC\Roboter\KRC\R1\System.

В рамках разработанного ПО используются следующие переменные:

COUNTER\_VARIABLE – необходима для использования пользовательских переменных.

COM\_CASEVAR – осуществляет выбор Case, в котором прописанные определенные операции (например выбор Базы, Инструмента и тд.)

COM\_FRAME – используется для выбора Базы, Инструмента.

COM\_VALUE – используется для выбора Скорости.

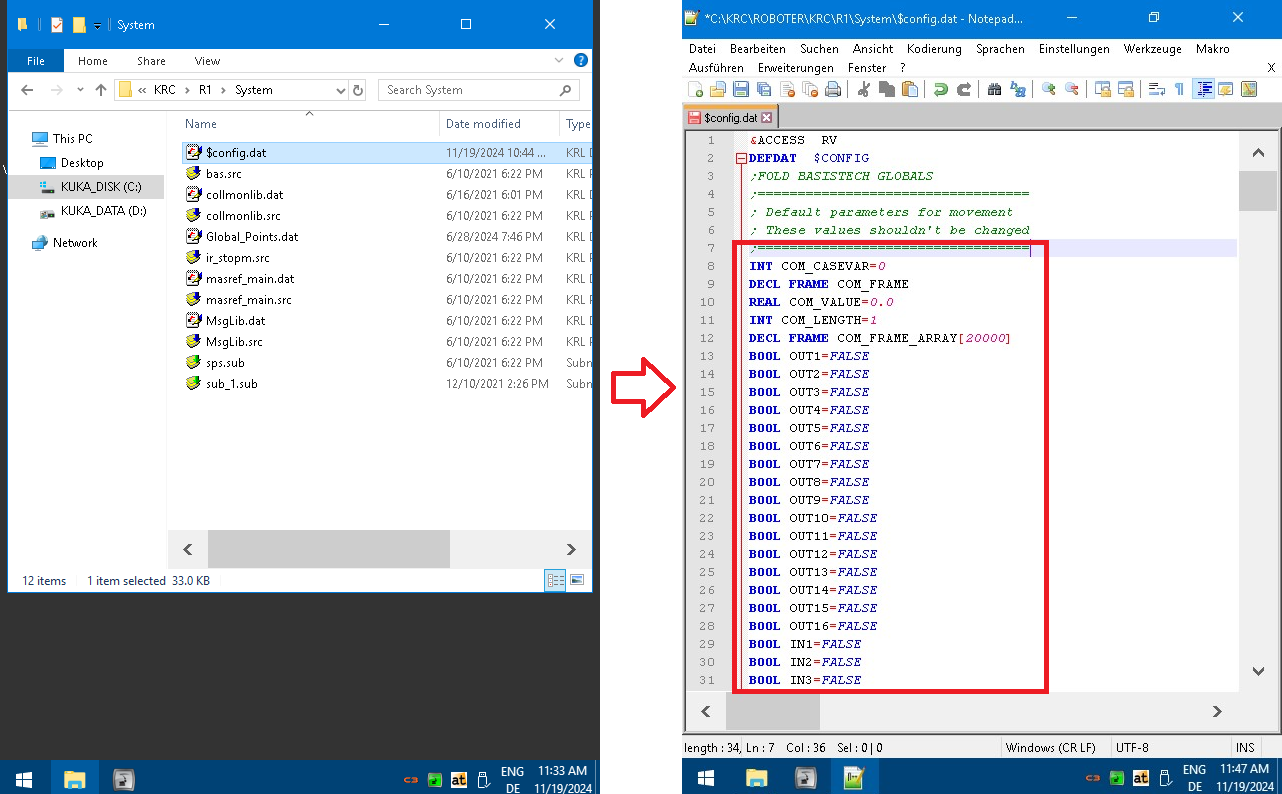
COM\_LENGTH – используется для определения количества переданных точек для движения манипулятора.

COM\_FRAME\_ARRAY – используется для передачи массива точек для движения манипулятора.

OUT1-16 – используется для управления цифровыми выходами.

IN1-16 – используется для считывания информации с цифровых входов.

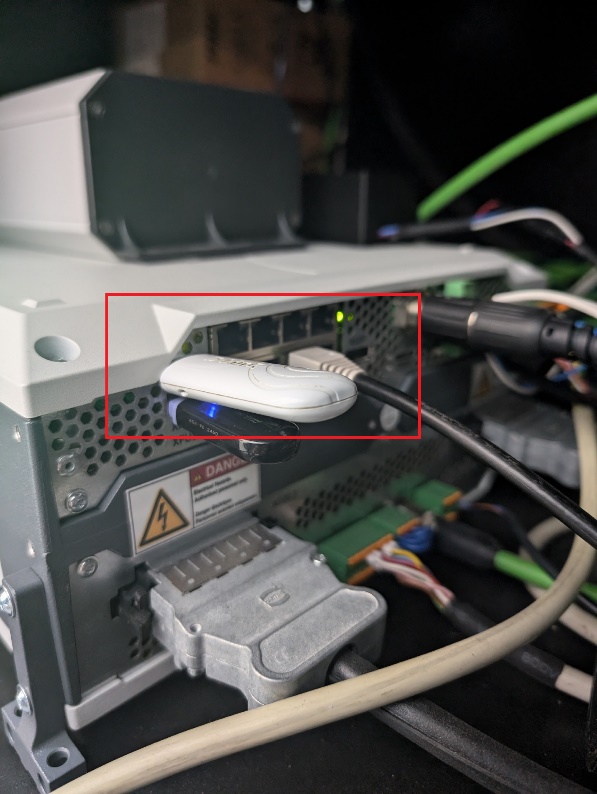
Переменные, добавленные в файл отображены на рис.1-14.



**Рис.1-14: Файл Config.dat**

## Добавление Wi-Fi модуля в контроллер манипулятора

Контроллеры манипуляторов Kuka не оснащаются Wi-Fi модулями, однако, к ним можно подключить внешние Wi-Fi адаптеры (тестировалось с адаптером D-Link DWA-160). Wi-Fi адаптер подключается в USB 3.0 interfaces на контроллере манипулятора, либо в XFUSB1, либо в XFUSB2. Пример подключения представлен на рис.1-15.



**Рис.1-15: Подключение Wi-Fi адаптера**

Далее из **ОС Windows** на контроллере манипулятора подключаемся к сети Wi-Fi в которой находится удаленное устройство.

***Рекомендуется выделить отдельный IP адрес для манипулятора в локальной сети, в которой он будет использоваться и добавить его в конфигурацию сетевых настроек ОС Windows.***

## Установка программы управления на манипулятор

Для взаимодействия удаленного устройства и манипулятора на последнем должна быть запущена программа, написанная на языке KRL, которая находится в режиме ожидания, пока не будет отправлена команда с внешнего устройства.

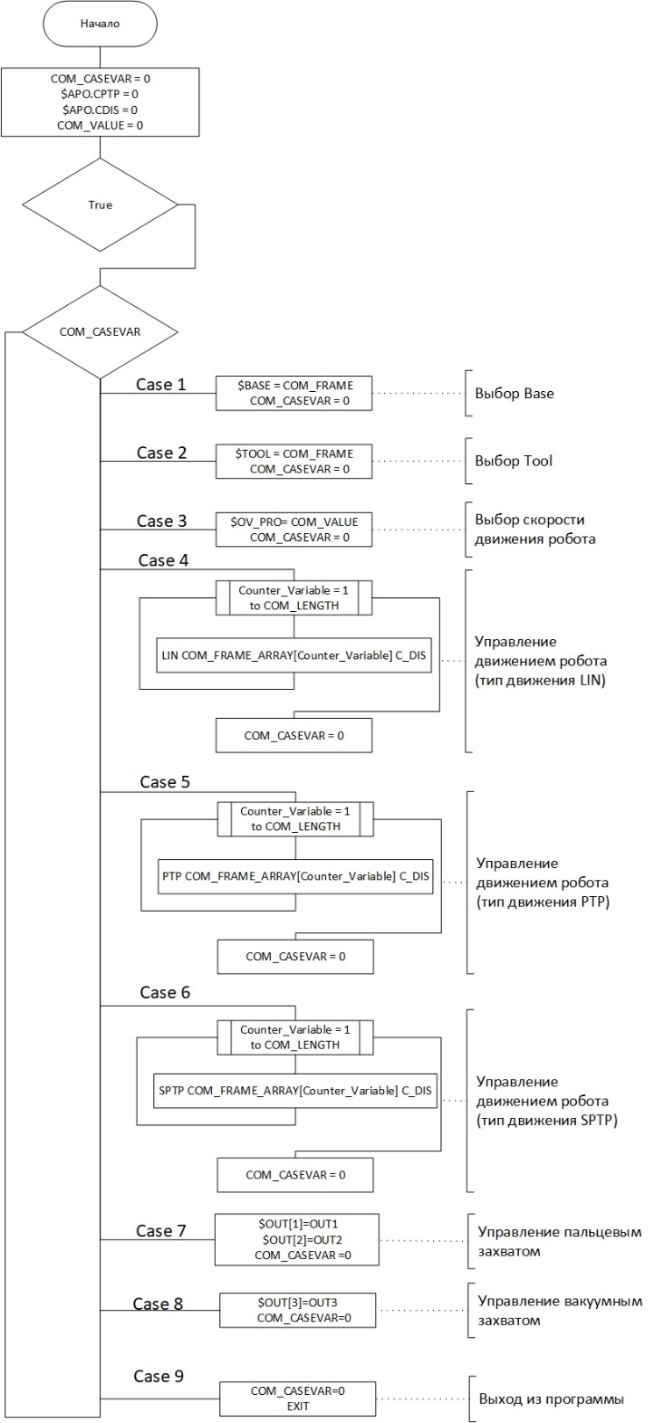
Такую программу можно написать, используя язык KRL на Kuka SmartPAD или загрузить готовую со своего устройства.

В рамках ПО используется программа Server.src, которая поставляется вместе с ПО.

Программу можно загрузить с помощью флеш-карты в любую папку контроллера манипулятора, предпочтительно загружать в **R1/Program**.

После загрузки программы манипулятор необходимо перезагрузить, как описывалось ранее.

Программа Server.src реализует алгоритм, показанный на рис.1-16.



**Рис.1-16: Алгоритм программы робота на языке KRL**

Взаимодействие с манипулятором организовано по принципу **Switch-Case**. Всего реализовано 9 модулей.

### Выбор BASE

Система координат **BASE** представляет собой декартову систему, описывающую рабочую зону манипулятора или объекта в рабочей зоне. Она определяется относительно системы координат **WORLD** и по умолчанию BASE[0] идентична ей.

Количество доступных пользовательских **BASE** зависит от контроллера KUKA и составляет до 32 штук. Изменение **BASE** позволяет перенести начало системы координат.

Выбор **BASE**, в которой будет работать манипулятор производится в ***Case 1***. BASE в манипуляторе хранятся в виде FRAME. FRAME – это структура данных, в которой содержится шесть параметров: {X, Y, Z, A, B, C}. Эти параметры соответствуют шести параметрам в представлении системы угловых координат: X, Y, Z, α, , , где α = A, = B, = C.

Изменить **BASE** возможно с помощью переменной $BASE\_DATA[номер] и указав номер, в котором будет работать манипулятор при отправке координат с удаленного устройства. Таким образом, функция считает FRAME, который соответствует базе с указанным номером и отправит его в переменную $BASE.

### Выбор TOOL

Алгоритм выбора инструмента находится в ***Case 2*** и повторяет выбор BASE, только для его корректной работы номер инструмента считывается с переменной $TOOL\_DATA, а записывается его FRAME в переменную $TOOL.

### Выбор скорости движения робота

Линейную скорость движения манипулятора можно установить в ***Case 3*** с помощью переменной $OV\_PRO (%). Для регулирования скорости движения манипулятора значение скорости передается в %, где максимальное значение равняется 100.

### Отправка координат движения

Для отправки координат для движения манипулятора необходимо отправить структуру данных FRAME в виде:

{FRAME: X значения, Y значения, Z значения, A значения, B значения, C значения},

где X, Y, Z – координаты относительно выбранной базы, A, B, C – углы поворота, A – угол вращения относительно оси X, B – угол вращения относительно оси Y, C – угол вращения относительно оси Z.

Значения углов и координат могут быть переданы как в формате float, так и в формате integer.

* + - 1. ***Case 4***

В ***Case 4*** возможна последовательная отправка массива с координатами точек для движения манипулятора. В качестве типа движения выбрано линейное **LIN**.

* + - 1. ***Case 5***

В ***Case 5*** возможна последовательная отправка массива с координатами точек для движения манипулятора. В качестве типа движения выбрано движение «от точки к точке» **PTP**.

* + - 1. ***Case 6***

В **Case 6** возможна последовательная отправка массива с координатами точек для движения манипулятора. В качестве типа движения выбран **SPTP**.

### Управление захватом

В манипуляторах Kuka для управления захватными устройствами используются цифровые выходы и входы (I/O interfaces). Например, в контроллере Kuka C5 Micro для этого используется интерфейс XG12.

Интерфейс ввода-вывода XG12 имеет 16 входов и 16 выходов:

• 16 входов, которые могут работать с датчиками;

• 16 выходов, которые могут работать с исполнительными устройствами.

Входы и выходы можно настроить группами по 8 портов в режимах высокого или низкого уровня считываемого сигнала:

• перемычка между контактами 1-2 переключает входы 1 - 8 в режим низкого уровня сигнала.

• перемычка между контактами 3-4 переключает входы 9 - 16 в режим низкого уровня сигнала.

• перемычка между контактами 5-6 переключает выходы 1 - 8 в режим низкого уровня сигнала.

• перемычка между контактами 7-8 переключает выходы 9 - 16 в режим низкого уровня сигнала.

По умолчанию: высокий уровень сигнала

Расположение пинов интерфейса XG12:

| **Номер пина** | **Описание выводов** |
| --- | --- |
| 1 | IN\_Config 1-8  Каналы настраиваются (в зависимости от положения перемычки) |
| 2 |
| 3 | IN\_Config 9-16  Каналы настраиваются (в зависимости от положения перемычки) |
| 4 |
| 5 | OUT\_Config 1-8  Каналы настраиваются (в зависимости от положения перемычки) |
| 6 |
| 7 | OUT\_Config 9-16  Каналы настраиваются (в зависимости от положения перемычки) |
| 8 |
| 9-39  Все нечетные номера выводов | IN 1 ... IN 16  Цифровые входы 1 - 16 |
| 10-40  Все четные номера выводов | OUT 1 ... OUT 16  Цифровые выходы 1 - 16 (выходной ток = 0,5 А) |

***Режим High-side***

Состояние для входов не определено для диапазона напряжений от 5 В до 11 В (диапазон перехода). Устанавливается либо состояние ON, либо состояние OFF.

* Сигнал Off / 0:

Состояние OFF для диапазона напряжений от -3 В до 5 В.

* Сигнал On / 1:

Состояние ON для диапазона напряжений от 11 В до 30 В.

Ток нагрузки на входе 3 … 11 мА

***Режим Low-side***

Состояние для входов не определено для диапазона напряжений от 7 В до 18 В (диапазон перехода). Устанавливается либо состояние ON, либо состояние OFF.

* Сигнал Off / 0:

Состояние OFF для диапазона напряжений от 18 В до 30 В.

* Сигнал On / 1:

Состояние ON для диапазона напряжений от 0 В до 7 В.

Ток нагрузки на входе -2 … -8 мА

В зависимости от захватного устройства управление им осуществляется с помощью одного или нескольких цифровых выходов, которые принимают значение True или False. Например, параллельный пальцевый пневматический захват может быть подключен к пневмораспределителю, который в свою очередь подключен к интерфейсу XG12 в OUT[1] и OUT[2], тогда в Case должно быть прописано управление этими выходами. Данные переменные являются системными и запрещены для редактирования в случае управления с внешнего устройства, но существует возможность их переопределения через другие переменные, которые ранее добавлены в файл config.dat. Таким образом предполагается использование ***Case 7*** для управления пальцевыми захватными устройствами и ***Case 8*** для управления вакуумным захватом.

В случае добавления новых захватных устройств необходимо добавить новый Case, который будет отправлять сигнал на соответствующие цифровые выходы.

### Использование программы совместно с ПЛК

Для управления передаваемыми от ПЛК номерами программ используется организационная программа **Cell.src**. Она всегда находится в папке **R1**. Как и любую обыкновенную программу, программу Cell также можно индивидуально настроить, однако при этом структура программы должна сохраняться.

В программе **Cell.src** используется аналогичная структура switch-case, где за выбор конкретного Case отвечает ПЛК. Следовательно, необходимо создать Case в программе **Cell.src**, в котором будет содержаться вызов программы **Server.src**.

После выполнения операции манипулятором необходимо вызвать ***Case 9***, который реализует алгоритм выхода из программы **Server.src**. ***Case 9*** позволяет передать в ПЛК информацию о том что манипулятор выполнил задачу. Например, с помощью системы дистанционного управления манипулятор захватил объект и поставил его на конвейер, тогда выход из программы может стать сигналом для запуска движения конвейера.

## Запуск программы на манипуляторе

Выполнять отладку программ рекомендуется в ручном режиме (T1 или Т2). В ручном режиме выполнение исполнительных команд, полученных от системы внешнего управления, может контролироваться оператором, что позволит избежать столкновения манипулятора с препятствием в случае не правильной работы системы внешнего управления (например, неправильно выбрана база/инструмент) или для отладки системы компьютерного зрения.

После отладки программы можно выполнить запуск **Server.src** в автоматическом режиме (AUT) или в режиме внешней автоматики (EXT). В этих режимах манипулятор будет выполнять команды сразу.

***Предупреждение:***

***При подключении с внешнего устройства программа на манипуляторе должна находится в цикле LOOP, в противном случае не будет выбрана Base и Tool и манипулятор будет идти в координаты, которые соответствуют $NULLFRAME.***

# Программный комплекс на удаленном устройстве

На удаленном устройстве в папке проекта должно находится два файла:

OpenShowVar.py – свободно распространяемый модуль, который позволяет использовать переменные и структуры данных управляемого манипулятора для чтения и записи. Более подробно про модуль можно почитать по ссылке: <https://ieeexplore.ieee.org/abstract/document/7349325>.

Kuka.py – библиотека команд для управления манипулятором с удаленного устройства.

## Библиотека Kuka.py

* **kuka.set\_base (base)**

Параметры:

base – номер базы от 1 до 32, предварительно откалиброванной на манипуляторе.

Результат:

Обращение к ***Case 1*** и выбор заданной базы как систему отсчета.

* **kuka.set\_tool (tool)**

Параметры:

tool– номер базы от 1 до 32, предварительно откалиброванной на манипуляторе.

Результат:

Обращение к ***Case 2*** и выбор заданного инструмента как систему отсчета.

* **kuka.** **set\_speed (value)**

Параметры:

value– скорость перемещения манипулятора в % от 0 до 100.

Результат:

Обращение к ***Case 3*** и задание скорости выполнения программы.

* **kuka.lin\_continuous (kuka, massiv)**

Параметры:

kuka – подключенный к системе дистанционного управления манипулятор.

massiv– массив координат в формате numpy.

Результат:

Обращение к функции massiv\_coord и формирование массива координат в формате, который может интерпретировать манипулятор. Полученный массив отправляется на манипулятор с помощью функции send\_Frame\_array. При этом в переменную COM\_LENGTH записывается длина массива и происходит обращение к ***Case 4***. Тип движения **LIN**.

* **kuka.ptp\_continuous (kuka, massiv)**

Параметры:

kuka – подключенный к системе дистанционного управления манипулятор.

massiv– массив координат в формате numpy.

Результат:

Обращение к функции massiv\_coord и формирование массива координат в формате, который может интерпретировать манипулятор. Полученный массив отправляется на манипулятор с помощью функции send\_Frame\_array. При этом в переменную COM\_LENGTH записывается длина массива и происходит обращение к ***Case 5***. Тип движения **PTP**.

* **kuka.sptp\_continuous (kuka, massiv)**

Параметры:

kuka – подключенный к системе дистанционного управления манипулятор.

massiv– массив координат в формате numpy.

Результат:

Обращение к функции massiv\_coord и формирование массива координат в формате, который может интерпретировать манипулятор. Полученный массив отправляется на манипулятор с помощью функции send\_Frame\_array. При этом в переменную COM\_LENGTH записывается длина массива и происходит обращение к ***Case 6***. Тип движения **SPTP**.

* **kuka.open\_grip()**

Параметры:

-

Результат:

Управление цифровыми выходами OUT[1] и OUT[2], к которым подключен пневмораспределитель и пальцевый захват. Открытие захвата и обращение к ***Case 7***.

* **kuka.close\_grip()**

Параметры:

-

Результат:

Управление цифровыми выходами OUT[1] и OUT[2], к которым подключен пневмораспределитель и пальцевый захват. Закрытие захвата и обращение **к *Case 7***.

* **kuka.vacuum\_on()**

Параметры:

-

Результат:

Управление цифровым выходом OUT[3], к которому подключен вакуумный захват. Включение захвата и обращение к ***Case 8***.

* **kuka.vacuum\_off()**

Параметры:

-

Результат:

Управление цифровым выходом OUT[3], к которому подключен вакуумный захват. Отключение захвата и обращение к ***Case 8***.

* **kuka.** **quit()**

Параметры:

-

Результат:

Обращение к ***Case 9*** и выход из программы **Server.src.**

* **kuka.** **read\_base()**

Параметры:

-

Результат:

Считывание значения системной переменной $BASE и запись её координат в переменные: base\_frame\_x, base\_frame\_y, base\_frame\_z, base\_frame\_A, base\_frame\_B, base\_frame\_C и общих координаты в переменную base\_frame.

* **kuka.** **read\_tool()**

Параметры:

-

Результат:

Считывание значения системной переменной $TOOL и запись координаты инструмента в переменные: tool\_frame\_x, tool\_frame\_y, tool\_frame\_z, tool\_frame\_A, tool\_frame\_B, tool\_frame\_C и общих координат в переменную tool\_frame.

* **kuka.** **read\_joint()**

Параметры:

-

Результат:

Считывание значения системной переменной $AXIS\_ACT и запись значений углов сочленений шестиосевого манипулятора в переменные: A1\_joint, A2\_joint, A3\_joint, A4\_joint, A5\_joint, A6\_joint и общих значений углов сочленений в переменную global\_position\_joint

* **kuka.** **read\_cartesian()**

Параметры:

-

Результат:

Считывание значения системной переменной $POS\_ACT и запись значений обобщеных координат в переменные: x\_cartesian, y\_cartesian, z\_cartesian, A\_cartesian, B\_cartesian, C\_cartesian и общих значений обобщеных координат в переменную global\_position. В случае использования манипулятора на передвижной платформе используются еще 6 обобщеных координат: E1\_cartesian, E2\_cartesian, E3\_cartesian, E4\_cartesian, E5\_cartesian, E6\_cartesian.

* **kuka.** **send\_Frame (arr, system\_variable="")**

Параметры:

arr– массив координат в формате numpy.

Результат:

Отправка в манипулятор значений точек движения в формате:

{FRAME: X значения, Y значения, Z значения, A значения, B значения, C значения},

где X, Y, Z – координаты относительно выбранной базы, A, B, C – углы поворота, A – угол, вращающийся вокруг оси X, B – угол, вращающийся вокруг оси Y, C – угол, вращающийся вокруг оси Z.

* **kuka.** **robot\_read(types)**

Параметры:

types – название переменной в формате “$Название переменной”.

Результат:

Чтение с манипулятора значения системной переменной, переданной в types и вывод результата на экран.

Функции lin\_continuous\_4, ptp\_continuous\_4, sptp\_continuous\_4, send\_Frame\_array\_4, send\_Frame\_4, massiv\_coord\_4 работают аналогично вышеописанным функциям, но используются для управления четырехосевым манипулятором Scara.

### ****Программирование удаленного управления****

**Общее для всех вариантов:**

Студенты должны использовать предоставленные библиотеки OpenShowVar.py и Kuka.py. Предполагается, что соединение с манипулятором уже настроено.

#### **Вариант 1: "Калибровщик"**

1. Напишите скрипт, который выполняет точное позиционирование манипулятора по заранее известным координатам.
2. **Алгоритм:**
   * Установите базу BASE[3] и инструмент TOOL[2].
   * Задайте низкую скорость движения (15%).
   * Отправьте манипулятор в точку с координатами {X: 450, Y: 150, Z: 300, A: 0, B: 90, C: 0} с использованием движения **PTP**.
   * После достижения точки, прочитайте и выведите в консоль фактические значения $POS\_ACT для проверки точности позиционирования.

#### **Вариант 2: "Инспектор"**

1. Напишите скрипт для сканирования зоны и сбора данных.
2. **Алгоритм:**
   * Установите скорость 40%.
   * Прочитайте и сохраните в текстовый файл значения углов в сочленениях ($AXIS\_ACT) и декартовых координат ($POS\_ACT) в 3-х различных точках рабочей зоны, между которыми манипулятор перемещается с помощью движения **LIN**.
   * В каждой точке программа должна делать паузу в 2 секунды перед снятием показаний.

#### **Вариант 3: "Сборщик"**

1. Напишите скрипт, имитирующий сбор детали с конвейера и ее укладку в тару.
2. **Алгоритм:**
   * Установите базу BASE[5] (координаты конвейера).
   * Используйте движение **PTP** для быстрого подхода к точке над деталью {X: 100, Y: 50, Z: 200, A: 0, B: 0, C: 0}.
   * Используйте движение **LIN** для точного спуска к точке захвата {Z: 95}.
   * Закройте захват (close\_grip()).
   * По траектории **LIN** поднимите деталь, переместите ее над тарой и опустите.
   * Откройте захват (open\_grip()).

#### **Вариант 4: "Сортировщик"**

1. Напишите скрипт, сортирующий объекты по двум зонам.
2. **Алгоритм:**
   * Установите высокую скорость (80%).
   * Реализуйте цикл, который 2 раза выполняет последовательность:
     + Захват объекта в точке {X: 200, Y: 0, Z: 100}.
     + Если это первая итерация, перенесите объект в Зону A {X: 300, Y: 200, Z: 150}.
     + Если это вторая итерация, перенесите объект в Зону B {X: 300, Y: -200, Z: 150}.
   * Используйте тип движения **PTP** для быстрых перемещений между зонами.

#### **Вариант 5: "Крановщик"**

1. Напишите скрипт для перемещения крупногабаритного груза.
2. **Алгоритм:**
   * Установите скорость 25%.
   * Включите вакуумный захват (vacuum\_on()).
   * Используя только движение **LIN** (для минимизации раскачки), переместите груз из точки {X: 500, Y: 0, Z: 400} в точку {X: -500, Y: 0, Z: 400}.
   * Выключите вакуумный захват (vacuum\_off()).
   * Вернитесь в исходную точку.

#### **Вариант 6: "Оператор"**

1. Напишите интерактивный скрипт, управляющий манипулятором с клавиатуры.
2. **Алгоритм:**
   * Скрипт должен в цикле ожидать ввода пользователя.
   * Реализуйте команды:
     + w/s - перемещение инструмента вперед/назад по оси X на 10 мм.
     + a/d - перемещение влево/вправо по оси Y на 10 мм.
     + q/e - подъем/опускание по оси Z на 10 мм.
     + o/c - открыть/закрыть захват.
     + r - вывести текущие координаты.
     + x - выход из программы и вызов kuka.quit().

#### **Вариант 7: "Тестировщик инструментов"**

1. Напишите скрипт для проверки калибровки разных инструментов.
2. **Алгоритм:**
   * Программа должна поочередно устанавливать инструменты TOOL[1], TOOL[2] и TOOL[3].
   * Для каждого инструмента манипулятор должен подходить к одной и той же тестовой точке {X: 0, Y: 0, Z: 400, A: 0, B: 0, C: 0} с использованием **PTP**.
   * После каждого подхода скрипт считывает и выводит на экран фактические координаты инструмента ($POS\_ACT), позволяя оценить точность калибровки.

#### **Вариант 8: "Палейнер"**

1. Напишите скрипт, который перемещает манипулятор по траектории прямоугольника в горизонтальной плоскости.
2. **Алгоритм:**
   * Установите скорость 30%.
   * Сформируйте массив из 4 точек, образующих квадрат со стороной 100 мм.
   * Отправьте массив на манипулятор и выполните движение по контуру квадрата с использованием типа движения **LIN**.
   * Программа должна вернуть манипулятор в исходную точку.

#### **Вариант 9: "Силовой"**

1. Напишите скрипт, ориентированный на работу с цифровыми входами/выходами.
2. **Алгоритм:**
   * Установите скорость 60%.
   * Напишите цикл, который ожидает сигнал на цифровом входе IN[1] (например, от датчика наличия детали).
   * Как только сигнал получен, манипулятор должен:
     + Подойти к месту забора детали.
     + Закрыть захват.
     + Активировать цифровой выход OUT[4] (например, сигнал "Деталь взята").
     + Переместить деталь в зону разгрузки.

#### **Вариант 10: "Автономный работник"**

1. Напишите комплексный скрипт, объединяющий несколько операций и завершающий работу.
2. **Алгоритм:**
   * Установите базу BASE[1] и инструмент TOOL[1]. Скорость 50%.
   * Выполните 3 полных цикла, каждый из которых включает:
     + Захват детали в точке А.
     + Перемещение детали в точку Б.
     + Возврат в исходное положение.
   * После выполнения всех циклов, прочитайте и запишите в файл итоговые значения $AXIS\_ACT (положение сочленений).
   * Корректно завершите работу с манипулятором, отправив команду выхода из Server.src.