Universal Robot Commissioning Guide

This document outlines basic instructions for configuration of the Universal Robot in the PFI AMG Robotic Palletising System.

28 April 2021

Table of Contents

[1. Program Transferring 3](#_Toc70506831)

[1.1 Configuration Backup 3](#_Toc70506832)

[1.2 Program Upload 3](#_Toc70506833)

[1.3 Program Backup 3](#_Toc70506834)

[1.4 Robot Emulator 3](#_Toc70506835)

[2. Robot Positioning 4](#_Toc70506836)

[2.1 Teach Pendant 4](#_Toc70506837)

[2.1.1 Absolute Coordinates 4](#_Toc70506838)

[2.1.2 TCP Movement Arrows 5](#_Toc70506839)

[2.1.3 Joint Movement 5](#_Toc70506840)

[2.2 Manual Drive 5](#_Toc70506841)

[3. Plane Configuration 6](#_Toc70506842)

[4. Program Structure 8](#_Toc70506843)

[4.1 Background 8](#_Toc70506844)

[4.1.1 Programming Movement 8](#_Toc70506845)

[4.1.2 Variable Assignment 8](#_Toc70506846)

[4.2 IO Mappings 8](#_Toc70506847)

[4.3 Program Overview 10](#_Toc70506848)

[4.3.1 Setup 10](#_Toc70506849)

[4.3.2 Operations 11](#_Toc70506850)

[4.3.2.1 PreTask 12](#_Toc70506851)

[4.3.2.2 MoveHome 12](#_Toc70506852)

[4.3.2.3 StartTask 12](#_Toc70506853)

[4.3.2.4 SetWaypoints 13](#_Toc70506854)

[4.3.2.5 SanitiseInputs 14](#_Toc70506855)

[4.3.2.6 MoveWaypoints 15](#_Toc70506856)

[4.3.2.7 FinishTask 16](#_Toc70506857)

[4.3.2.8 PrintThread 16](#_Toc70506858)

# Program Transferring

To transfer between the robot controller and computer a USB stick and the corresponding [magic file](https://www.universal-robots.com/download/?filters%5b%5d=98765&query=)[[1]](#footnote-1) (downloaded from the Universal Robots [software page](https://www.universal-robots.com/download/)[[2]](#footnote-2)) are required. Magic files are just bash scripts which contain the file transfer logic. The general process is as follows.

1. Download the magic file
2. Save this magic file on to the USB stick
   1. The remainder of the step will change based off which operation is being performed. See succeeding headings for more details.
3. Open the control box and plug in the USB
4. A red “! USB !” warning sign will appear on the screen
5. The file will automatically run. Do not remove the USB while this is in process, as it could corrupt the operating system.
6. A green “<- USB” sign will appear on the screen. It is now okay to remove the USB.

Files are transferred to the root directories of both the USB and robot. Transfers do not delete any auxiliary files.

## Configuration Backup

It is recommended to backup the configuration files to protect against operating system corruption. This should be the first transfer done.

Magic file: [urmagic\_configuration\_files.sh](https://www.universal-robots.com/download/software-e-series/support/magic-files/magic-file-backup-configurations-files/)

## Program Upload

Uploading programs copies all robot program files (.urp, .script, . installation etc) on the USB to the control box. Clear the USB of all files except those which are being transferred.

Magic file: [urmagic\_upload\_programs.sh](https://www.universal-robots.com/download/software-e-series/support/magic-files/magic-file-upload-all-programs-from-usb-to-robot/)

## Program Backup

Downloading programs copies all the robot program files from the control box on to the USB. Clear the USB of all files before downloading.

Magic file: [urmagic\_backup\_programs.sh](https://www.universal-robots.com/download/software-e-series/support/magic-files/magic-file-backup-all-programs/)

## Robot Emulator

This is an offline [simulator](https://www.universal-robots.com/download/?filters%5b%5d=98759&query=) where changes may be made to the robot program. It exists as a virtual machine and must be run with software like [vmware](https://www.vmware.com/au/products/workstation-player/workstation-player-evaluation.html).

# Robot Positioning

The robot can be moved through the teach pendant or manually. Either method should be done through the Move tab, which is displayed as follows. The Robot, Feature and TCP panes describe the position of the robot. The Feature pane details which feature the values in the TCP pane are relative to. The selected View feature is referenced to the rendered image in the Robot Pane. Other selectable features are the Base, Tool and custom lines, points and panes.

Graphical user interface

Description automatically generated

Figure Teach pendant move tab

## Teach Pendant

The teach pendant offers a controlled and precise way to move the robot.

### Absolute Coordinates

To send the robot to absolute coordinates enter the desired values into the TCP pane. The X, Y, Z variables are polar coordinates and RX, RY, RZ are the tool rotation in radians. Once the values are entered, hold down Auto to move to the position as prompted. In this project scope given coordinates are most likely relative to the stack. If this is the case the relevant custom stack feature should be selected through the Feature pane.

**Example.** Drive to the coordinates relative to Stack A.

|  |  |  |
| --- | --- | --- |
| 1. Select stack\_a and enter coordinates. RY is 3.1415 as the tool should be rotated 180º around the Y axis (green). | 2. Hold down the auto button, and the robot will drive to this position. | 3. The robot is now placed in the new position. |
|  | Graphical user interface  Description automatically generated | Graphical user interface, application  Description automatically generated |
| Figure 2 Driving progression from setting (left) to moving (middle) to verifying (right) | | |

### TCP Movement Arrows

The TCP may be moved relative to the selected feature along one of its six degrees of freedom with the movement arrows. These are located in the TCP Position pane which gives polar movement, and TCP Orientation Pane which gives rotational movement. With this method a fixed distance cannot be provided.

**Example.** Match directions to arrows

Diagram

Description automatically generated with low confidence

Figure Move arrorrws

Move horizontally right along the stack face Move horizontally left along the stack face

Move vertically up the stack face Move vertically down the stack face

Move directly away from the stack face Move directly towards the stack face

### Joint Movement

Specific joints may be moved with the arrows and input boxes within the Move Joints pane. While this is not as useful as any of the other methods it does provide some feedback on closeness to joint limits, as indicated by the black arrow. Each joint may move between -360º and +360º by default. This behaviour may be changed through the installation menu.

## Manual Drive

Manual drive offers a fast way to position the robot in a general area. It is more rugged and harder to get exact than the teach pendant. To enable manual drive click and hold the Freedrive button on the Move Joints pane or push and hold the physical button behind the teach pendant. Gently guide the robot to its position. It is easiest to move joints which are on the same axis. Complex movements which affect multiple joints require more effort through shear force, and are also clunkier. To this end it is easiest to apply pressure near individual joints. Avoid applying force to the tool appendage.

# Plane Configuration

Custom features may be defined within the installation tab. These are linked to the selected installation file. They can be points, lines or planes. The robot may be moved relative to these. In this application the stack faces at both gantry positions are represented as planes. This is required as the pallet may not be perfectly centred and thus will appear different relative the base after a change in gantry position. Features can be seen under the Features menu item.

Graphical user interface

Description automatically generated

Figure Stack plane features

These planes may need to be reconfigured to align them with the vision system coordinates. To enter the configuration wizard select the feature and then click Teach this plane. This opens a welcome page outlining the steps for creation/alteration.

Graphical user interface

Description automatically generated

Figure Plane reconfiguration options

**Steps:**

1. ***Welcome Screen***

This shows the sequence of points to move the TCP to in the following three steps. These are each physically located on the calibration frame. It is evident that the calibration frame is mirrored from the wizard – this is explained in step 4.

Graphical user interface, application

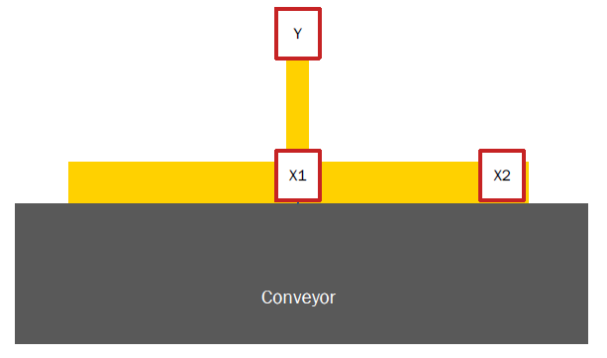
Description automatically generated

Figure Plane points virtual and physical

1. ***Origin***

This is located on the calibration frame at point X1. Move the robot to this position. It is easiest to do so manually. Ensure that the printer head is flush against the bar. Note that the rotation of the tool is irrelevant here as the rotation of the overall plane is determined from the cartesian points. Follow the procedure pictured as follows.

Graphical user interface, Word

Description automatically generatedGraphical user interface

Description automatically generatedGraphical user interface, Word

Description automatically generated

Figure Setting the first waypoint

1. ***Positive X-Axis***

This is located on the calibration frame at point X2. Move the robot to this position as in step two.

1. ***Direction of Positive Y-Axis***

This is located on the calibration frame at point Y. This step breaks from the wizard mold slightly. The welcome screen pictures it directly above point 2 defined in the previous step, yet on the frame it is above the origin. This is okay as a direction vector is being defined and since axes are definitively orthogonal all changes in the already defined X are ignored[[3]](#footnote-3).

1. ***Accept Plane***

Visually check this plane and create. To verify correctness navigate to the Move tab and select the created feature. Use the TCP Movement Arrows to traverse the plane and ensure the TCP moves parallel or orthogonally depending on the chosen direction. Move to known positions on the calibration frame and check that values in the TCP pane correspond to physical values.

# Program Structure

After going through the background knowledge this section outlines at a high level how the robot program works. AMG robotic palletizing project scope is assumed knowledge.

## Background

### Programming Movement

The robot moves by passing its TCP coordinate system through a series of waypoints with six degrees of freedom. These are encoded as a vector of which denotes the transformation. Points detail Cartesian transposition. The values detail the rotation. The following waypoint and image shows the effect of waypoint vector . The TCP (blue dot) indicates how the waypoint is shifted. The tool reference coordinates have been flipped around the base y-axis through and now the and unit vectors have been inverted with respect to the base.

A picture containing chart

Description automatically generatedA picture containing text, clock

Description automatically generatedr

Figure Example pose vector

Movements through waypoints can be done in reference to the predefined base and TCP or alternatively the custom features defined in the installation. **This is static and cannot be done through a variable defined at runtime**. This places a limitation in the programming stage for this application, as Stack A or Stack B cannot be dynamically selected. For this reason the robot program always uses the base as its reference.

### Variable Assignment

The program variables are not scoped, and all are considered global. Before the program runs these are all recognized by the compiler and initialized to a null value unless otherwise specified. This enables variables to be safely referenced before explicitly initialized within the code.

## IO Mappings

Table Input Mappings

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Inputs** | **Alias** | **Type** | **Action** | **Description** |
| GPbi[0] | output\_init | Bool | Auto-Init | PLC signal to initialize the robot |
| GPbi[1] | output\_stop | Bool | Stop-Program | PLC signal to stop the program |
| GPbi[2] | start\_continue | Bool | Start-Program | PLC signal to start the program, with task continue flag |
| GPbi[3] | start\_retry | Bool | Start-Program | PLC signal to start the program, with task retry flag |
| GPbi[4] | cancel\_home | Bool | - | PLC signal to stop the current task, and move home |
| GPbi[5] | gantry\_in\_pos\_a | Integer | - | PLC input detailing whether gantry is locked in pos a |
| GPbi[6] | gantry\_in\_pos\_b | Integer | - | PLC input detailing whether gantry is locked in pos b |
| GPii[0] | task | Integer | - | PLC signal to being the task number |
| GPii[1] | X1 | Integer | - | Vision system left box x position |
| GPii[2] | X2 | Integer | - | Vision system middle box x position |
| GPii[3] | X3 | Integer | - | Vision system right box x position |
| GPii[4] | Y | Integer | - | Vision system maximum of layer y positions |
| GPii[5] | Z | Integer | - | Vision system maximum of layer z positions |
| GPii[6] | Z\_MIN | Integer | - | Vision system maximum of stack z |

Table Output Mappings

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Outputs** | **Alias** | **Type** | **Actions** | **Description** |
| GPbo[0] | moving\_home | Bool | - | Output flag detailing whether currently moving home |
| GPbo[1] | homed | Bool | - | output flag whether the robot is sitting in a home position |
| GPbo[2] | task\_active | Bool | - | output flag whether a task is currently active |
| GPbo[3] | task \_done | Bool | - | output flag whether the last task has been completed |
| GPbo[4] | printing | Bool | - | output flag whether printing is active |
| GPio[0] | current\_task | Integer | - | output detailing the current task number |

## Program Overview

At a high level the program follows the succeeding flow.

Diagram

Description automatically generated

Figure High level process flow

### Setup

**Init variables** assigns values to the chosen variables. This occurs before anything else. Values are retained from the last program run if the keep from last flag is True. A default value is substituted if this flag is set and this memory is not available. Default expressions may also be supplied when the flag is False.

Table variables defined within the init start procedure

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Description** | **Keep From Last** | **Default Expression** |
| last\_task | Describes the previous task complete | True | 0 |
| print\_enable | Flag to block printing when False | True | False |
| print1 | Whether print area 1 is active | False | False |
| print2 | Whether print area 2 is active | False | False |
| print3 | Whether print area 3 is active | False | False |

**Before Start** sequence runs once before the program enters its main loop. The variables defined here provide customisability to the operations.

Table variables defined within the before start procedure

|  |  |  |
| --- | --- | --- |
| **Variable** | **Description** | **Expression** |
| clearance | Print head clearance to stack face | 0.04 (CONFIGURE) |
| ramp\_dist | Distance to accelerate over before printing | 0.1 (CONFIGURE) |
| trig\_width | The spread around the target area which printing may be triggered | 0.003 (CONFIGURE) |
| trig\_L2R\_dist | How far away from print area to trigger (left to right) | 0.01 (CONFIGURE) |
| trig\_R2L\_dist | How far away from print area to trigger (right to left | 0.057 (CONFIGURE) |
| stack\_rx | How the TCP coordinates should be rotated around stack x coords | 0 (MAY CONFIGURE) |
| stack\_ry | How the TCP coordinates should be rotated around stack y coords | -d2r(180) (MAY CONFIGURE) |
| stack\_rz | How the TCP coordinates should be rotated around stack z cords | 0 (MAY CONFIGURE) |
| box\_to\_pa\_xy | Vision system coordinates offset to print area center | p[0,0,0,0,0,0] (CONFIGURE) |
| stack\_a\_to\_ref | Vision system stack a plane adjustment to robot reference | p[0,0,0,0,0,0] (CONFIGURE) |
| stack\_b\_to\_ref | Vision system stack b plane adjustment to robot reference | p[0,0,0,0,0,0] (CONFIGURE) |
| print\_width | Width of the printed portmark | 0.04 |
| printer\_width | Width of the printer head | 0.125 |
| TE\_L2R | Task enumeration left to right | 1 |
| TE\_R2L | Task enumeration right to left | 2 |
| pa\_x\_clear | Physical separation between centre of portmark and printer head when leading print nozzle is ready to start | (print\_width+printer\_width)/2 |
| runover | How far to travel from the print area. Gives room to adequately accelerate before starting a layer. | pa\_x\_clear+ramp\_dist |
| tasks\_equal | Flag indicating whether the current task is same as last | task=current\_task |
| tasks\_printing | Flag indicating whether the last task was in the process of printing | task<>0 & current\_task <> 0 and print\_enable |
| restart\_task | Flag indicating whether a task should be restarted | start\_continue & not tasks\_equal & tasks\_printing |
| resume\_task | Flag indicating whether a task should be resumed | start\_continue & tasks\_equal & tasks\_printing |
| print\_enable | Flag indicating whether printing can be triggered | False |
| sp | Setpoint denoting the position of the robot when the program begins. | get\_actual\_joint\_positions() |

These variables and their combinations are summarised in the following diagram. This pictures the scenario where the printer is traversing from right to left. Between runover and pa\_x\_clear is the acceleration phase. This ensures that when the head is at pa\_x\_clear it is ready to begin shooting. The triggering variables are with respect to leading edges, and so will have to be offset by pa\_x\_clear to align with centre points.

**Diagram

Description automatically generated  
 Figure x. Variables diagram**

### Operations

**Robot Program** runs the continuous main loop to define the business logic. The flow of sub-procedures is shown in the succeeding figure.

|  |  |  |
| --- | --- | --- |
| Check that the gantry is locked into a single position | Diagram  Description automatically generated |  |
| Set the stack reference  Set clearance position |
| **Cancel home branch** | **Left to right, or right to left branch** |
| Retract to the home position  & set the homed flags | Signal to PLC a new task has started |
| **Print Thread**  Check whether the printer is within the printing range if printing flag is set | Calculate waypoints for current layer |
| Check that waypoints make physical sense, potentially error |
| Set printing flag  Move across waypoints  Remove print flag |
| Signal to PLC the current task is complete, wait for acknowledge and reset outputs |

#### PreTask

* ***Lines 01-04:*** Get the correct reference frame based off the gantry crane position, adjusted with stack to reference variable defined within the before task procedure.
* ***Lines 05-07:*** the CLEAR position is defined.
  + ***Lines 05:*** Get the current position relative to the reference, assign to vision\_pos\_ex
  + ***Lines 06:*** Adjust vision\_pos\_ex z coordinate to z\_min with clearance
  + ***Lines 07:*** Define CLEAR position as vision\_pos\_ex transposed to base coordinates



Table useful variables defined within PreTask

|  |  |
| --- | --- |
| **Variable** | **Description** |
| reference | Relevant stack reference frame |
| CLEAR | Base referenced position which is clear of the nearest stack element |

#### MoveHome

* ***Lines 01-03:*** Clear the relevant flags as per PLC interface
* ***Lines 04-10:*** Move to the home position, uninterrupted
  + ***Lines 05:*** set the moving home flag
  + ***Lines 06, 07:*** move linearly to the home position. A joint move is faster here, but less controlled and may move in the negative z direction to touch the stack.
  + ***Lines 08, 09, 10:*** Set the homed flag, and remove the moving home flag



#### StartTask

* ***Lines 01-02:*** Flags associated with the PLC interface



#### SetWaypoints

* ***Lines 01-13:*** Established movement order along the stack plane. The vision system always orders coordinates from left to right. If the robot is tasked left to right then this is correct. If the robot is tasked right to left then it must move to the third coordinate first, and the first coordinate third.
  + ***Lines 01:***  The middle coordinate will always be the middle box, set this accordingly.
  + ***Lines 02, 08:*** Choose a branch based on the task direction.
  + ***Lines 04, 10:*** Set a x-direction modifier . This will be multiplied against future x-direction adjustments to keep them standardised with the coordinate system, irrespective of the task direction. Herein the direction of movement is referred to as xm-direction.
    - *E****.***g. A way point must be placed meters ahead of the robot position in its current direction. When moving left to right this is with the coordinate system and so can be simply added. When moving right to left this is against the coordinate system and must be subtracted.

This is not a convenient representation in code, as this adjustment is occurring in multiple locations. It is more convenient to have this set to a reusable variable as follows to abstract the coordinate reversal.

* + ***Lines 06, 07, 11, 12:*** Choose the box ordering based on direction.
  + ***Lines 08, 13:***  Set the offset to trigger before reaching the print area. This is required by the printer.r
* ***Lines 14-17:*** This provides a static adjustment from the given print coordinates, to the desired print location. If the vision system gives the top left of the print area, then this would be adjusting to the center of the print area.
* ***Lines 18-21:*** Define the vision system waypoints which the robot will traverse along
  + ***Lines 19:*** PA1\_EN describes the print area one entry position. It is the first print area transposed negative runover in the xm-direction and position clearance in the z-direction. Rotation variables force the correct printer orientation relative to the stack coordinates.
  + ***Lines 20:*** PA3\_EX describes the print area three exit position. It is the third print area transposed positive runover in the xm-direction and position clearance in the z-direction. Rotation variables again force the correct printer orientation relative to the stack coordinates.
  + ***Lines 21:*** APPROACH describes the PA1\_EN approach. It is the same xy position, however moved to the static z position This allows a large blend radius between the entry to keep some speed.
* ***Lines 22-24:*** This moves all defined points to the base coordinate system. This is important as movements cannot be done relative to a variable position defined at runtime.



Table useful variables defined within SetWaypoints

|  |  |
| --- | --- |
| **Variable** | **Description** |
| xm | Abstracts the direction of x movement to this multiplier |
| trigger\_offset | How far from the print area before the print bit should be set |
| PA1\_EN | Entry position for the first print area, relative to the base |
| PA3\_EX | Exit position for the third print area, relative to the base |
| APPROACH | Approach position for PA1\_EN, relative to the base |

#### SanitiseInputs

TBD

#### MoveWaypoints

* ***Line 02-27:*** This section defines how the robot will setup and position before the print sequence. Decisions are based off the robot current position, and special startup cases. All movements may be cancelled early with the cancel\_home flag.
  + ***Lines 02-08:*** Beginning at the home position, move directly to the approach and then entry
  + ***Lines 09-12:*** Resume task startup case, so no repositioning is required. Clear this flag to avoidr repeated triggers.
  + ***Lines 13-18:*** Have recently finished a left to right, and now are asked to do a right to left. In this scenario the robot is not directly off the stack face at any point during positioning. As there is no danger of touching the stack it can directly reposition to the entry point.
  + ***Lines 18-27:*** All other scenarios. Retract to a clearance position. Move to the approach, and then the entry.
* ***Lines 29-33:*** This defines the robot print sequence.
  + ***Lines 30:*** Set the print enable flag
  + ***Lines 31,32:*** Already being position at the start, move to the final position until reached or cancel\_home is set
  + ***Lines 33:*** Clear the print enable flag



Table useful variables defined within MoveWaypoints

|  |  |
| --- | --- |
| **Variable** | **Description** |
| print\_enable | Flag which enables/disables print operations |

An example traversal is as follows.

|  |  |  |
| --- | --- | --- |
| **Task 1: Left to right layer 1**  Start from home, so we move  APPROACH, PA1\_EN -> PA3\_EX {lines 02-08} | Diagram  Description automatically generated | Diagram  Description automatically generated |
| **Task 2: Right to left layer 2**  Clear of the stack, so simply move  PA1\_EN -> PA3\_EX {lines 13-18} |
| **Task 3: Left to right layer 3**  Not clear of the stack, so move CLEAR, APPROACH, PA1\_EN -> PA3\_EX {lines 19-27} |
| **Task 4: Right to left layer 4**  Clear of the stack, so simply move  PA1\_EN -> PA3\_EX {lines 13-18} |

#### FinishTask

* ***Lines 02-04:*** Notify the PLC with required flags that a task has been completed
* ***Lines 05:*** Wait until the PLC has acknowledge, or exit early if cancel\_home is set
* ***Lines 06, 07:*** Remove flags



#### PrintThread

* ***Lines 02:*** Checks that the print flag is set before continuing
* ***Lines 03:*** Gets the current TCP position relative to the vision plane
* ***Lines 04-06:*** Boolean checks to see if the TCP is ready to print
  + This checks the absolute difference between the current X and the desired print X, and checks that it is within the trig\_width range
* ***Lines 07, 08:*** If in any of the print areas, set the configurable output to a HI pulse for printing



The following diagram shows the triggering dimensions. The variable trig\_R2L\_dist provides physical spacing between the trigger and when printing should occur. This is made relative to center points by adding pa\_x\_clear to get trigger\_offset. Tolerance area is denoted as trig\_width.

Diagram

Description automatically generated

1. [https://www.universal-robots.com/download/?filters[]=98765&query=](https://www.universal-robots.com/download/?filters%5b%5d=98765&query=) [↑](#footnote-ref-1)
2. <https://www.universal-robots.com/download/> [↑](#footnote-ref-2)
3. This may be empirically proven. Relative to the base define a plane PN1 with points {1: (500, 0, 800), 2: (500, 250, 800), 3:(500, **0**, 1050)} and plane PN2 with points {1: (500, 0, 800), 2: (500, 250, 800), 3:(500, **250**, 1050)}. PN1 and PN2 are equivalent . [↑](#footnote-ref-3)