11: The KUKA Robot Programming Language

TOPIC 1: Movement

The KUKA robot can move from point A to point B in three main ways.

1. PTP – Point-to-Point – Motion along the quickest path to an end point. This motion requires the programmer to "teach" one point.

2. LIN - Linear – Motion at a defined velocity and acceleration along a straight line. This motion requires the programmer to "teach" one point. The robot uses the point defined in the previous move as the start point and the point defined in the current command as the end point and interpolates a straight line in between the two points.

3. CIRC – Circular – Motion at a defined velocity and accerlation along a circular path or a portion of a circular path. This motion requires the programmer to "teach" two points, the mid-point and the end point. Using the start point of the robot (defined as the end point in the previous motion command) the robot interpolates a circular path through the mid-point and to the end point.

TOPIC 2: Inputs and Outputs

Inputs and Outputs (I/O) are the same concept as in PLCs.

INPUTS – An input is something (digital or analog) coming from another system and is read in and used to make decisions. Inputs cannot be changed by the robot is represent the state of something external to the robot such as whether a sensor is on or off. In the robots our inputs are defined from 33 through 40. These inputs can be set to any number but the external inputs that are numbered 0 through 7 are reflected in the programming language as 33 through 40. Therefore to refer to physical input 0 in the program the syntax is $IN[33]. Physical input 4 would be $IN[37].

OUTPUTS – Outputs can be changed by the robot but can also be monitored. The numbering is the same as the inputs. Physical output 0 is output 33 in the program. The sytax is $OUT[33] for output 33. The syntax to change the state of the output is $OUT[33]=TRUE to cause physical output 0 to turn on and $OUT[33]=FALSE to cause physical output 0 to turn off.

PULSED OUTPUTS – An output can be turned on for a short (or defined) period of time easier than just turning the output on, waiting for a time, and then turning the output off. An input cannot be pulsed because it cannot be altered by the robot. The syntax to accomplish this is PULSE($OUT[33],TRUE,0.5) or more generically PULSE($OUT[#],*state*,*time*) where *state* can be TRUE or FALSE and *time* is the time to pulse the output in seconds. The above example would turn on physical output 0 for 0.5 seconds. This time can range from 0.012 seconds to 2^31 seconds in increments of 0.1 seconds.

TOPIC 3: Execution Control

The following are different ways to control the execution of your program.

1. If statements – An if statements checks a condition and executes code if the condition is true and may execute code (if written) if the condition is false. The syntax is as follows.

IF *conditional* == TRUE THEN

Whatever code you want to execute when the conditional is TRUE

ELSE

Whatever code you want to execute when the conditional is FALSE

ENDIF

For example, if you had a switch connected to physical input 0 the following code might be used.

IF $IN[33]==TRUE THEN

the code written here would execute when the switch was on.

ELSE

the code written here would execute when the switch was off.

ENDIF

The ELSE statement is optional and if not used should not be entered in. In other words if in the example above nothing should happen if the switch was off the following code could be used.

IF $IN[33]==TRUE THEN

the code written here would execute when the switch was on.

ENDIF

For those of you who are familiar with programming, there is no "ELSEIF" option. However, you can nest the IF statements within each other.

2. Switch statements – A switch statement (in other languages it is called a case statement) is commonly used when a variable can have many values instead of just on and off. For example, if a variable had the name counter_variable and it could attain values of 10, 20, 30, 40, or 50 the following code could be used to execute different code base on the value of the variable.

SWITCH counter_variable

  CASE 10

    code that should execute when counter_variable equals 10.

  CASE 20

    code that should execute when counter_variable equals 20.

  CASE 30

    code that should execute when counter_variable equals 30.

  DEFAULT

    code that should execute when counter_variable doesn't equal any of the above cases.

ENDSWITCH

3. For loops – The For loop is a command that allows the programmer to execute a piece of code a certain number of times while incrementing through a variable. For example if a programmer wanted to execute a set of code 50 times while incrementing a variable in steps of 2 the following code could be used.

FOR Counter_Variable = 1 to 100 STEP 2

  code to execute every time through the loop

ENDFOR

The STEP is optional and without the command it defaults to 1.

4. While loops – Instead of executing a set of code a set number of times, a While loop can be used to execute a piece of code while a condition remains true or false. For example if a robot should move back and forth while an input remains on the following code could be used.

WHILE $IN[35]==TRUE

  code to execute while the input remains on

ENDWHILE

5. Repeats loops - A While loop does something while a condition remains true or false, but a repeat loop does something until a conditions becomes true or false. For example, a robot could be asked to move back and forth until a condition is met.

REPEAT

code to be executed until input 40 turns off.

UNTIL $IN[40]==FALSE

6. Endless loops – Many times, it is the desire of the programmer that the robot does the same task over and over again endlessly. In order to accomplish that we use the LOOP command. This causes code between LOOP and ENDLOOP to execute without end.

LOOP

code to execute endlessly

ENDLOOP

TOPIC 4: Variables

In a basic program (MODUL PROGRAM) there is a INI line pre-written in a new program. Above this INI line is the area in which variables are declared or given name and definition. The following are allowable variable types.

1. Integer (numbers without a decimal point such as 1, 233, 143, 4365) – Syntax: INT *variable_name*
2. Real (numbers with a decimal point such as 1.2, 33.45, 3.14) - Syntax: REAL *variable_name*
3. E6POS (variable representing a point in space and robot orientation) – Syntax: E6POS *variable_name*

The E6POS variable consists of 6 variables representing the point in cartesian space and the orientation of the arm at that point. Because of this the programmer can reference an E6POS variable in several ways.

- variable_name.x would refer to the x value of the point in space
- variable_name.y would refer to the y value of the point in space
- variable_name.z would refer to the z value of the point in space
- variable_name.a would refer to the rotation around the z axis in space
- variable_name.b would refer to the rotation around the y axis in space
- variable_name.c would refer to the rotation around the x axis in space

Oftentimes a programmer will want to save the current position of the robot to an E6POS variable. This is done with the $POS_ACT command as follows.

If point_in_space is the variable name the programmer would type… point_in_space = $POS_ACT and the current position of the robot would be saved to the variable in real time.

Additionally, when using variables, many operators are required and can be grouped into three categories.

1. Relational Operators

- Check to see if equal to: ==
- Check to see if not equal to: <>
- Check to see if less than: <
- Check to see if less than or equal to: <=
- Check to see if greater than: >
- Check to see if greater than or equal to: >=

2. Logic Operators

- NOT
- AND
- OR
- EXOR (exclusive or)

3. Arithmetic Operators

- Multiplication *
- Additiona +
- Subraction -
- Division /

TOPIC 5: Other Topics (include wait and waitfor here)

Timers – Timers are also available to the programmer for uses such as timing the amount of time that occurs between two inputs coming on. There are 16 timers (1-16) and there are three commands available for each timer. Shown here are the commands for timer 1. If any other timer is used just replace the 1 with the timer number.

- $TIMER_STOP[1]=FALSE This command starts the timer timing. Just like the button on your stopwatch that starts timing.
- $TIMER_STOP[1]=TRUE This command stops the timer timing. Just like the button on your stopwatch that stops timing.
- $TIMER[1] This is the place where the time is stored in milliseconds. A programmer can set this value to zero by typing $TIMER[1]=0 or use conditional statements based on the value. The value can also be used in a mathematical equation such as Distance = Rate * Time to determine either distance or rate whichever is not known. We will use this in the lab to determine the speed of the conveyor belt by solving for Rate = Distance / Time using two sensors placed along the conveyor a known distance apart.

Velocity Command – The linear velocity of the robot can be set by using the $VEL.CP variable. We will also use this in the lab to set the robot to move at the same speed as the conveyor. An example would be if we wanted the robot to move at 0.5m/s we would type $VEL.CP=0.5. Note that this variable is always in units of m/s.

Wait Commands – There are three different commands that the programmer can use to cause the program to freeze.

- WAIT FOR – This command causes the program to stop until a condition is met. An example would be WAIT FOR $IN[35]. This would cause the program to stop until $IN[35] was true.
- WAIT SEC – This command causes the program to stop for a certain amount of time. An example would be WAIT SEC 3.2. This would cause the program to freeze for 3.2 seconds.
- HALT – The HALT command causes the robot program to stop until restarted by an operator.