JOpenShowVar: some possible high-level applications

F. Sanfilippo, L. I. Hatledal and H. Zhang
Department of Maritime Technology and Operations, Aalesund University College
Postboks 1517, 6025 Aalesund, Norway
{fisa, laht, hozh}@hials.no

M. Fago
IMTS S.r.L. Company
Taranto, Italy
massimiliano.fago@gmail.com

K. Y. Pettersen
Centre for Autonomous Marine Operations and Systems
Department of Engineering Cybernetics, Norwegian University of Science and Technology
7491 Trondheim, Norway
kristin.y.pettersen@itk.ntnu.no

I. Some other possible high-level applications

In addition to these methods, some other high-level functions can be implemented by the user on top of the *JOpenShowVar* communication protocol. The following subsections provide the user with some guidelines that can be used to implement such methods. It should be noted that these possible high-level methods are not included in the *JOpenShowVar* library simply because their implementation depends on how the user declares the desired variables and the corresponding procedures on the KRC side.

Writing the robot's joint angles: A useful application that can be achieved consists of setting the joint angles, all at once. Let $q_d = [\theta_1, \theta_2, ..., \theta_{n_j}]^T$, with n_j being the number of joints, be the final desired joint configuration of the robot. q_d can be stored in a local *KRLAxis* variable. Remotely, on the KRC side, a corresponding *Axis* variable structure should be created with the same name in the predefined global system data list $CONFIG.DAT so that it can be accessed from the KRL program that will control the robot. By using the *writeVariable* method the entire desired configuration for the robot can be written at once as shown in the Algorithm 1 sketch box for a six DOFs robot.

Writing the robot's end-effector position and orientation: Let $p_d = [x, y, z, \phi, \gamma, \psi]^T$ be the desired robot's end-effector position and orientation. p_d can be stored in a local *KRLPos* variable. Remotely, on the KRC side, a corresponding *Pos* variable structure should be created with the same name in the predefined global system data list $CONFIG.DAT so that it can be accessed from the KRL program that will control the robot. By using the *writeVariable* method the desired robot's end-effector position and orientation can be written at once as shown in the Algorithm 2 sketch box.

Writing the robot's path (joint space): In a possible application scenario, it is often necessary to deal with paths defined in the joint space. Let $Q = [q_1, q_2, ..., q_n]^T$, where n is the number of desired joint configurations, which together make out the desired path in the joint space, and where $q_i \in \Re^{n_j}$. Since the new method *writeVariable* cannot handle arrays, each desired joint configuration has to be sent as a string with the *sendRequest* method. Basically, the *sendRequest* function is iteratively

```java
KRLAxis qd = new KRLAxis("MYAXIS");
//MYAXIS is defined manually in $CONFIG.DAT
qd.setA1ToA6(80, 10, -10, 20, 35, 32);
try (CrossComClient client = new CrossComClient("localhost", 7000)) {
  client.writeVariable(qd);
}
```

Algorithm 1: Writing the robot's joint angles, Java side.

```java
KRLPos pd = new KRLPos("MYPOS");
//MYPOS is defined manually in $CONFIG.DAT
pd.setXToZ(100, 12, 30);
pd.setAToC(-20, 25, 53);
try (CrossComClient client = new CrossComClient("localhost", 7000)) {
  client.writeVariable(pd);
}
```

Algorithm 2: Writing the robot's end-effector position and orientation, Java side.

```
public void writeJointsPath(List<double[]> jointsPath)  throws IOException {
  try (CrossComClient client = new CrossComClient ("localhost", 7000)) {
    int i = 1;
    for (double[] d : jointsPath) {
      client.sendRequest (new Request (0, "MYE6ARRAY [" + i++ + "]" "{A1" + d [0] + "A2" + d [1] + ", A3"
          + d [2] + "A4" + d [3] + "A5" + d [4] + "A6" + d [5] + "}"));
    }
  }
}
```

Algorithm 3: Generate a path (joint space), Java side.

```
DECL INT ROW    ;array index declaration
  FOR ROW = 1 TO 512
    PTP MYE6ARRAY[ROW]
  END FOR
```

Algorithm 4: Generate a path (joint space), KRC side.

used to update a global array of E6AXIS on the KRC side. For each iteration, the new desired joint configurations are actuated with a PTP command. A possible use-case is suggested for a six DOFs robot. The Algorithm 3 sketch box shows a possible Java code, while The Algorithm 4 shows a possible implementation on the corresponding KRL side.

Writing the robot's path (Cartesian space): Similarly, it can be useful to generate paths in the Cartesian space. Let $P = [p_1, p_2, ..., p_n]^T$, where n is the number of desired Cartesian configurations, which together make out the desired path in the Cartesian space and where $p_i \in \Re^6$. Also in this case, the *sendRequest* function is adopted. In particular, this time the *sendRequest* iteratively updates a global array of POS on the KRC side. For each iteration, the new desired Cartesian configurations are actuated with a PTP or LIN command. A possible use-case is suggested here. The Algorithm 5 sketch box shows the possible Java code, while The Algorithm 6 shows the possible implementation on the corresponding KRL program.

Setting binary outputs: In order to provide the possibility of setting binary outputs that can be used to open or close valves or control a gripper, another top level method could be added. It is not possible to set an output directly, but it can be done in a SPS loop cycle by means of global variables as shown in the Algorithm 7 sketch box:

```
public void writeCartesianPath(List<double[]> cartesianPath)  throws IOException {
  try (CrossComClient client = new CrossComClient ("localhost", 7000)) {
    int i = 1;
    for (double[] d : cartesianPath) {
      client.sendRequest (new Request (0, "MYPOS [" + i++ + "]" "{X" + d [0] + "Y" + d [1] + ", Z" + "}"))
          ;
    }
  }
}
```

Algorithm 5: Generate a path (Cartesian space), Java side.

```
DECL INT ROW    ;array index declaration
  FOR ROW = 1 TO 512
    PTP MYPOS[ROW]
  END FOR
```

Algorithm 6: Generate a path (Cartesian space), KRC side.

```
SPS LOOP
...
$OUT[1] = OUTPUT_VARS[1]
...
$OUT[n] = OUTPUT_VARS[n]
...
SPS END LOOP
```

Algorithm 7: Setting binary outputs, KRC side.