

**Listing of “Ideas For Further Explorations” provided in the book
“Projects Guide For ROBOTIS ENGINEER”**

Ideas for further explorations (IFFE 2.1):

2.1.1.a: Modify “SBwA_PositionControl_1.task3” so that only a simple Time Delay is used when Pose_3 is called, but make this Time Delay around 300-400 ms only. Then readers can see how the 2XL430 deals with the situation when it is “ordered” to go somewhere else when it is in the midst of its trip going towards the “original” Goal Position, but has not gotten there yet!

2.1.1.b: Modify “SBwA_PositionControl_1_mm.task3” so that MaxVel and MinVel information can be searched for and collected for a time frame defined by two Hi-Res Timer counters Timer1 and Timer2 (for example Timer1=2950 ms and Timer2=2450 ms).

Ideas for further explorations (IFFE 2.2):

2.1.2.a: This ROBOTIS web link provides formulas to compute velocity timings for various Position Control modes (<http://emanual.robotis.com/docs/en/dxl/x/2xl430-w250/#profile-velocity112>). In the current “RECTANGULAR” case, ($t_1 = 0$) because (Profile Acceleration = 0), ($t_2 = 64 * \Delta\text{Position} / \text{Profile Velocity} = 64 * (3072-2048) / 75 = 874$ ms for the transition from VT=75 to VT=0). However, the following two screen captures at run-time indicated that this transition happened around 899-901 ms instead!

854 -75 2096 -74 214 0	859 -75 2092 -74 2134
866 -75 2083 -74 2128	870 -75 2080 -74 2123
877 -75 2072 -74 2114	880 -75 2068 -74 2110
888 -75 2059 -74 2101	891 -75 2055 -74 2099
899 0 2048 -74 2090	901 0 2048 -74 2086
909 0 2048 -74 2079	912 0 2048 -74 2076
920 0 2048 -73 2071	923 0 2048 -72 2067

Upon further experimentations with different values for Profile Velocity (e.g. 100, 500 and 1000), these results indicated that this coefficient should be “66” instead of “64”. Thus, at least, for the author’s set up:

$$t_2 = 66 * \Delta\text{Position} / \text{Profile Velocity} = 66 * (3072-2048) / 75 = 901 \text{ ms.}$$

It will be instructional to see whether the readers find the same constant or perhaps another completely different constant!

Ideas for further explorations (IFFE 2.3):

2.1.6.a: The interested reader can explore the similarities and differences between `MOTION OFFSET` and `ADJUSTED OFFSET` in the example code “`SBwA_RC_MotionPlay_JointOffset_200_264.tsk3`”. The reader can try **either/or** for each option or apply **both** options.

2.1.6.b: Taking advantage of the contiguity of Memory Addresses for Joint Offsets, the example program “`SBwA_RC_MotionPlay_JointOffset_32.tsk3`” applies **randomized offsets** to all 4 servos of the SBwA robot using a **32-bit** (i.e. Double-Word) Offset to write **Two Offsets per each Custom Write command** (i.e. first to Servos 1 & 2 together, then to Servos 3 & 4 together). The usage of the **Offset Control Parameter** (Address 199) is also illustrated.

Ideas for further explorations (IFFE 2.4):

2.2.1.a: Modify “`SBwA_PositionControl_1.py`” so that only a simple Time Delay is used when `Pose_3` is called and set this Time Delay to 300-400 ms. Then readers can see how the 2XL430 deals with the situation when it is “ordered” to go somewhere else when it is in the midst of its trip going towards the “original” Goal Position, but has not gotten there yet!

2.2.1.b: Modify “`SBwA_PositionControl_1_mm.py`” so that `maxVel` and `minVel` information can be searched for and collected for a time frame defined by two Hi-Res Timer counters `Timer1` and `Timer2` (for example `Timer1=2950 ms` and `Timer2=2450 ms`).

2.2.1.c: “`SBwA_PositionControl_1_mm.py`” can be modified so that each Servo 1 to 4 and the CM-550 have more distinct “names”:

```
hc = DXL(200)
s1 = DXL(1)
s2 = DXL(2)
s3 = DXL(3)
s4 = DXL(4)
```

Then “`DXL(200).write16(74,3000)`” can be written as “`hc.write16(74,3000)`” and `DXL(1).goal_position(arms_close_position)` as “`s1.goal_position(arms_close_position)`” for easier “code reading”- without any noticeable change in run-time performance.

Ideas for further explorations (IFFE 2.5):

2.2.2.a: This ROBOTIS web link provides formulas to compute velocity timings for various Position Control modes (<http://emanual.robotis.com/docs/en/dxl/x/2xl430-w250/#profile-velocity112>).

*In the current “RECTANGULAR” case, ($t_1 = 0$) because (Profile Acceleration = 0), ($t_2 = 64 * \Delta\text{Position} / \text{Profile Velocity} = 64 * (3072-2048) / 75 = 874$ ms for the transition from $V_T=75$ to $V_T=0$). However, Fig. 2.46 showed that this transition likely happened around 892-905 ms instead!*

Upon further experimentations with different values for Profile Velocity (e.g. 100, 500 and 1000), the author’s experimental results indicated that this coefficient should be “66” instead of “64”. Thus, at least, for the author’s set up:

$$t_2 = 66 * \Delta\text{Position} / \text{Profile Velocity} = 66 * (3072-2048) / 75 = 901 \text{ ms.}$$

It will be instructional to see whether the readers find the same constant or perhaps another completely different constant!

Ideas for further explorations (IFFE 2.6):

2.2.6.a: The interested reader can also review the example program `SBwA_RC_MotionPlay_JointOffset_200.py` as an example of setting Motion Offsets without having to use the MOTION SubTool.

2.2.6.b: Taking advantage of the contiguity of Memory Addresses for Joint Offsets, the example program “`SBwA_RC_MotionPlay_JointOffset_32.py`” applies **randomized offsets** to all 4 servos of the SBwA robot using a **32-bit** (i.e. Double-Word) Offset to write **Two Offsets per each Custom Write command** (i.e. first to Servos 1 & 2 together, then to Servos 3 & 4 together). The usage of the **Offset Control Parameter** (Address 199) is also illustrated. Unfortunately, this program yielded a run-time Overflow Error whenever `etc.write32()` was used (for reasons unknown to the author). Thus, the author rewrote it for a **16-bit offset** value instead, as “`SBwA_RC_MotionPlay_JointOffset_16.py`” which performed as intended at run-time. The reader can compare the run-time performance of “`SBwA_RC_MotionPlay_JointOffset_16.py`” vs its TASK equivalent “`SBwA_RC_MotionPlay_JointOffset_32.task3`” from Section 2.1.6.

Ideas for further explorations (IFFE 3.1):

3.2.4.a: Line 132 in Fig. 3.41 defined *Line_Average* as a simple Arithmetic Average:

$$\text{Line_Average} = (\text{Line_Deg_1} + \text{Line_Deg_2}) / 2.0 \quad \text{[a]}$$

We can also write the same formula but in a different form **[b]**:

$$\text{Line_Average} = 0.5 * \text{Line_Deg_1} + 0.5 * \text{Line_Deg_2.0} \quad \text{[b]}$$

which provides a different connotation: i.e. we treated “**present**” and “**future**” information with **equal importance**.

But we can emphasize “present” information over “future” information if we use form **[c]**:

$$\text{Line_Average} = 0.6 * \text{Line_Deg_1} + 0.4 * \text{Line_Deg_2.0} \quad \text{[c]}$$

Or we can emphasize “future” information over “present” information if we use form **[d]**:

$$\text{Line_Average} = 0.4 * \text{Line_Deg_1} + 0.6 * \text{Line_Deg_2.0} \quad \text{[d]}$$

We can implement even a more drastic emphasis using the multipliers **0.7** and **0.3** instead. The reader is encouraged to try out these ideas to see if the robot would perform differently at runtime for the same Color Track.

Ideas for further explorations (IFFE 3.2):

3.2.5.a: Use ROBOTIS Marker 6 to stand for a STOP command.
