以模型化基礎設計進行自動化系統的設計、模擬、及實現

By Bill Chou, Product Marketing Manager, MathWorks

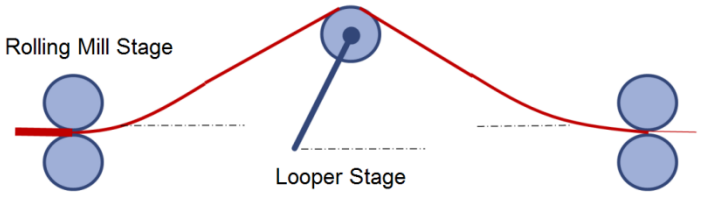
傳統的自動化工程師透過在與真實機器連結的可程式邏輯控制器(programmable logic controllers, PLCs)上執行控制演算法，來進行演算法的實現及測試。這種方式主要的缺點包含昂貴、困難、或首次於機器上測式控制策略的危險性。較妥善的作法是先利用機器模型的模擬來設計及測試控制系統。由於修正設計瑕疵的成本及困難度將隨著流程的進行而增加，模擬的使用可以讓許多問題在開發過程的初期即被揭露，減少在流程後端發現瑕疵的機會。在隨後控制器連接到真實機器進行測試及驗證時，工程師便可以對於系統實際表現更具信心。

模型化基礎設計

許多工程師體認到傳統控制設計途徑上的限制，開始使用模型化基礎設計(Model-Based Design)來開發及模擬代表著控制系統動態的系統層級模型。該模擬模型將控制策略與機器中的機械、電氣、液壓等要素連結。模擬幫助工程師執行案例測試、以及在開發流程中及早發現設計和整合上的錯誤。當一項專案由設計進行至執行階段，您可以從控制器模型產生結構式文件編程語言程式碼，並將其作為附加指令(Add-On Instruction，AOI) 匯入至 RSLogix™，以減少手動編寫程式碼可能發生的錯誤。透過模型化基礎設計，可以更容易地對控制策略進行變更或更新，因為您可以很快速地更新模型、重新執行測試、重新產生結構式文件編程語言程式碼、將更新後的 AOI 匯入至 RSLogix™。

鋼鐵軋延機系統建模

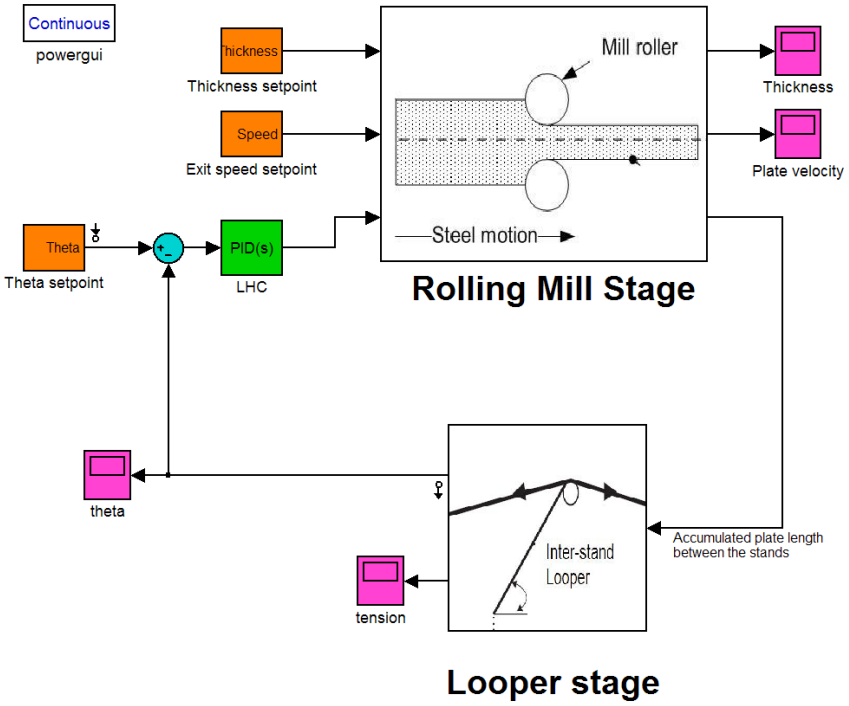
為了更清楚地了解模型化基礎設計實際的應用，我們以一個鋼鐵軋延機系統的控制策略開發為例。這個鋼鐵軋延機從扁鋼胚生產出單一厚度的鋼片，在軋延的過程當中，通常會由輥輪在每個階段對鋼片壓縮。在軋延的過程中間，會加入一個張力控制器(looper)來維持鋼片張力的穩定，避免鋼片撕裂或鬆弛。



(圖 1) 軋延機流程示意圖

建立單一步驟的受控體模型

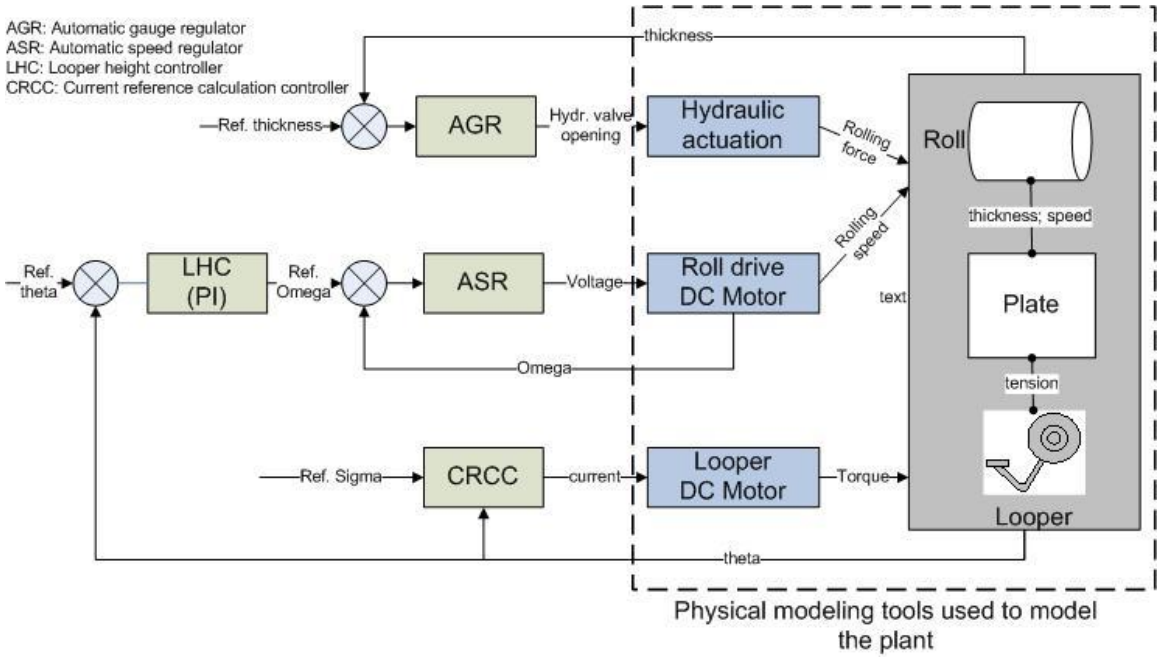
完整多步驟系統建立可以從建立由輥輪與張力控制器組成的較小的單一步驟受控體模型開始。在軋延階段，一個液壓致動器被用來對輥輪施加壓縮鋼帶的壓力。這個以 DC 馬達產生的軋延扭矩負責控制軋延速度。您可以在 Simulink 建立輥輪的機械、電氣、以及液壓元件的模型。張力控制器、張力控制器前的鋼片、及張力控制器之後的鋼片成為模型中以接頭連結的三個主體。



(圖 2)) Simulink 模型中的軋延與張力控制步驟示意

單一步驟的控制器建模

接下來要進行的是設計與建立單一步驟的輥輪控制器模型。圖 3 說明了一個使用了四個補償器的典型設計。自動標準規格調節器(automatic gauge regulator, AGR)補償器命令液壓閥開啟來產生輥輪壓縮的施力以控制鋼帶厚度。自動速度調節器(automatic speed regulator, ASR)補償器控制了 DC 馬達的電壓，產生軋延扭矩來控制鋼帶的速度。張力控制器的高度控制(looper height control, LHC)補償器依期望的材料張力決定輥輪的旋轉速度。最後，電流參考計算控制器(current reference calculation controller, CRCC)補償器管理傳送到張力控制器 DC 馬達的電流，以決定張力控制器的位置來維持材料張力。以上所有的環節都是成對的；AGR 補償器控制的液壓致動器控制了鋼帶的厚度與速度，而 LHC 與 ASR 補償器負責維持必要的張力及鋼帶速度。



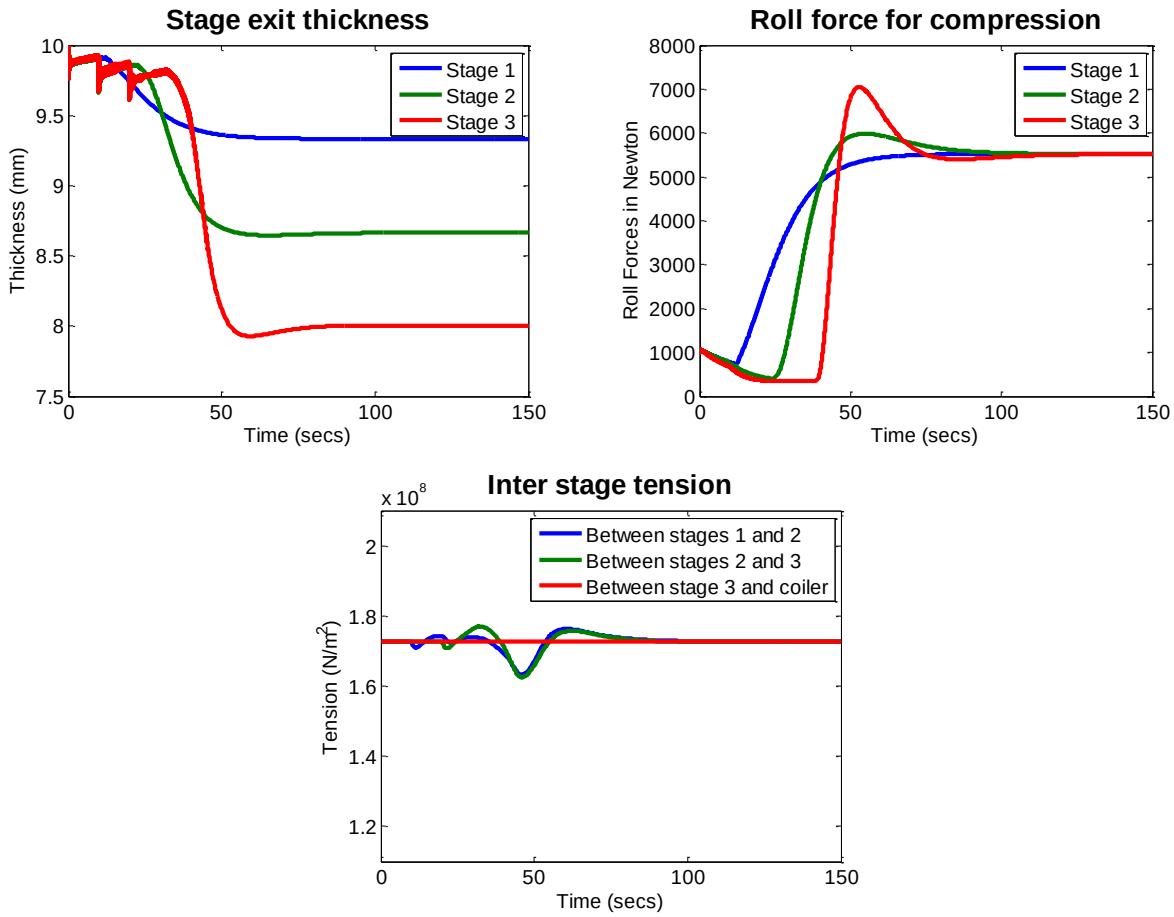
(圖 3) 多環路控制架構

下一個步驟要來調整 AGR 及 ASR 補償器。您首先須將非線性的受控體模型線性化，接著自動調整並推導出補償器相關係數。接下來，您可以透過執行單一步驟控制器與非線性受控體模型模擬來驗證推導出的相關係數數值。在這個案例，LHC 補償器模型是以等比例的整數(proportional integral, PI)控制器建立。使用 PID 調整工具可依據時間與頻率域的需求自動計算 PID 增益。最後，以推導出的相關係數執行非線性模擬來測試及驗證鋼軋機的整體設計。

建立一個多環流程

單一步驟的鋼軋機與張力控制器模型可以作為多環流程模型中的元件被重複使用。在這個階段，您也可以加入額外的元件，像是不同階段機器之間的大規模維護或輸送延遲。模擬可以幫助您看到在流程三階段中出現的流程變數。每個階段厚度與速度的均達到設定點，以在離開第三個階段時(圖 4 以紅色標示)製作出符合期望厚度的鋼片及達到期望產出。在不同工作站出現擾亂鋼片張力的因素也已有效地消除。

受控體模型的使用有兩個目的。第一，該模型已經過線性化，而線性化的模型被用來調整補償器。第二，以完整、非線性受控體模型進行模擬，進行不同狀況的測試，幫助了解不同情境所帶來的影響。



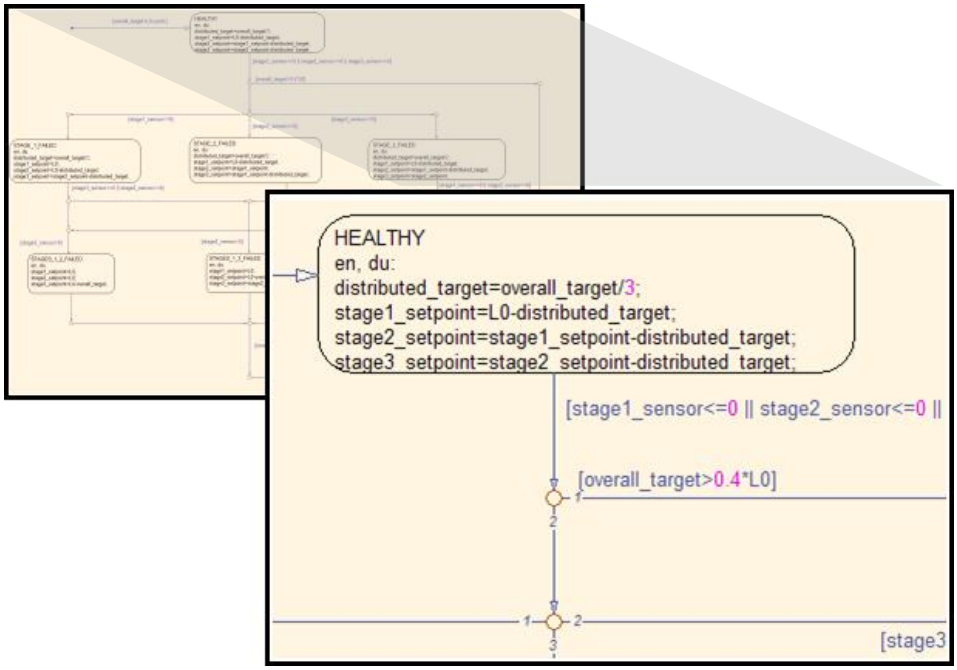
(圖 4) 流程變數的模擬結果。(a) 在離開第三階段時達到降低整體厚度的目標。(b) 降低整體厚度的目標被平均地分配在三個階段。(c) 擾亂鋼片張力的因素已被消除。

錯誤偵測邏輯的設計及驗證

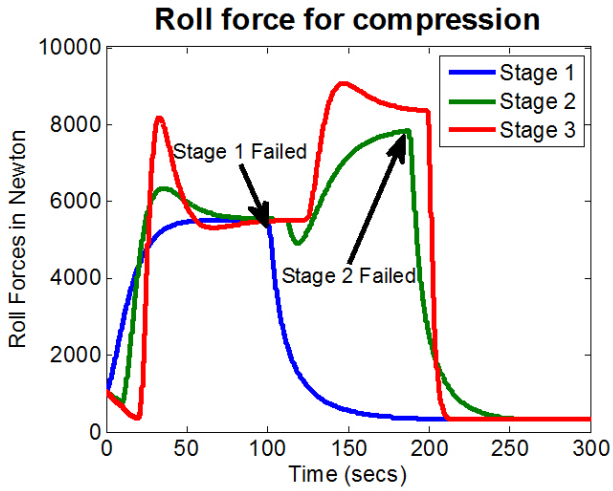
除了設計補償器，您還需要設計控制器內的錯誤偵測邏輯來監控與管理感測器及致動器。當控制器偵測到錯誤，必須將錯誤隔離在特定的元件，並且在不干涉正常操作的前提下嘗試處理錯誤。如果錯誤嚴重到無法恢復，則控制器必須讓機器安全地關閉。

這個範例著重於在液壓閥偵測錯誤及採取正確行動的錯誤偵測邏輯(圖 5)。特別是邏輯會分配整體厚度降低的目標於多環流程中各階段的個別厚度設定點。當其中一個階段的液壓壓縮出現錯誤，其他兩個階段的厚度降低的設定點將重新計算以達成整體厚度降低的目標。

您可以透過人工的方式將錯誤放進模型來對邏輯進行測試。圖 6 顯示錯誤容許邏輯的模擬結果。當其中一個階段發生錯誤，負責管理的控制器會檢查其負載能否被分配到其他正常運作的階段。如果可以，新的厚度降低設定點會被傳送至各階段的 AGR。如果無法進行分配，則暫停鋼片移動來停止流程。



(圖 5) 一部份於 Stateflow 執行的錯誤容許設定點分配



(圖 6) 模擬結果顯示系統在遇到階段錯誤時如何進行修復：第二與第三階段會在第一階段出現錯誤時提供補償。當第二階段發生錯誤時，由於整體厚度降低的目標無法只透過第三階段達成，因此系統會關閉。

於 PLC 執行結構式文件編程語言程式碼

在邏輯上，自動產生有組織的文字為設計驗證之後的下一個步驟。它可以幫助消除手動編寫程式碼所產生的錯誤，並且幫助確保最終的結構式文件編程語言程式碼可以在 Allen-Bradley® PLC 產生與模擬結果非常相似的數值的結果。重複利用也使用於模擬及測試的同一個模型，您可以自動產生控制器演算法的結構式文件編程語言程式碼，並接著匯入至 RSLogix™，來編譯與載入程式碼至 RSLogix™ Emulate 或真正的 Allen-Bradley PLC。圖 7 為從錯誤偵測與調節邏輯產生的 IEC 61131 結構式文件編程語言程式碼。這些產生的結構式文件編程語言程式碼具備良好的註解，且可以很輕鬆地追溯回模型。Simulink 可程式控制器文本產生器 (Simulink PLC Coder)也可以產生一個測試台，來驗證模擬結果是否與 RSLogix™ Emulate 結果相符。

```
        (* During 'HEALTHY': '<S1>:1' *)
        IF ((stage1_sensor <= 0) OR (stage2_sensor <= 0)) OR (stage3_sensor <= 0) THEN
            (* Transition: '<S1>:140' *)
            IF overall_target > (0.4 * L0) THEN
                (* Transition: '<S1>:141' *)
                (* Transition: '<S1>:142' *)
                is_c2_Setpoint := 2;
                (* Entry 'NO_REDUCTION': '<S1>:123' *)
                rtb_stage1_setpoint := L0;
                rtb_stage2_setpoint := L0;
                distributed_target := L0;
            ELSE
                (* Transition: '<S1>:143' *)
                IF stage1_sensor <= 0 THEN
                    (* Transition: '<S1>:144' *)
                    (* Transition: '<S1>:146' *)
                    is_c2_Setpoint := 6;
                    (* Entry 'STAGE_1_FAILED': '<S1>:119' *)
                    distributed_target := overall_target / 2;
                    rtb_stage1_setpoint := L0;
                    rtb_stage2_setpoint := L0 - distributed_target;
                    distributed_target := rtb_stage2_setpoint - distributed_target;
                ELSIF stage3_sensor <= 0 THEN
                    (* Transition: '<S1>:145' *)
                    (* Transition: '<S1>:148' *)
                    is_c2_Setpoint := 8;
                    (* Entry 'STAGE_3_FAILED': '<S1>:118' *)
                    distributed_target := overall_target / 2;
                    rtb_stage1_setpoint := L0 - distributed_target;
                    rtb_stage2_setpoint := rtb_stage1_setpoint - distributed_target;
                    distributed_target := rtb_stage2_setpoint;
                ELSIF stage2_sensor <= 0 THEN
                    (* Transition: '<S1>:147' *)
                    is_c2_Setpoint := 7;
                    (* Entry 'STAGE_2_FAILED': '<S1>:117' *)
                    distributed_target := overall_target / 2;
                    rtb_stage1_setpoint := L0 - distributed_target;
                    rtb_stage2_setpoint := rtb_stage1_setpoint;
                    distributed_target := rtb_stage1_setpoint - distributed_target;
                ELSE
                    guard := 1;
                END_IF;
            END_IF;
        ELSE
            guard := 1;
        END_IF;
```

Comments with references to Stateflow Chart

(圖 7) 透過 Simulink 可程式控制器文本產生器產生包裝於附加指令 (AOI) 的 IEC 61131 結構式文件編程語言程式碼

總結

模型化基礎設計幫助工程師們模擬整個系統，並在開發周期及早對其設計進行驗證。自動從相同的模型產生結構式文件編程語言程式碼則有助於消除手動編寫程式碼可能發生的錯誤。當工程師將模型以模擬進行驗證並接著自動執行，他們可以更具信心地期待初次控制器與真實機器連結時整個系統的表現將如同預期。