Tyler Hardy

twhardy

ECE 6420 Section 400

Takehome #2

Description:

PID loop controllers make use of feedback to produce a stable output signal. PID controllers utilize a proportional, integral, and derivative calculation to provide a correctable error value. The proportional portion of the controller is based on a gain factor that is provided to the system. This will attempt to provide linear control of the error signal. The integral term in the controller performs a total summation of past error values to provide a compensation value based on past system performance. The derivative term is a way to predict the future error values in the system based on rate of change of the system. These three terms are combined to produce a total error signal. This error value is used to drive the plant to the desired stable state. Research has indicated that fuzzy logic may be used to provide a more accurate control loop.

References:

-CLIPS Reference Manual, Volume 1: Basic Programming Guide
-CLIPS Reference Manual, Volume 2: Advanced Programming Guide
-Head First C by O'Reilly
-ECE6420 Lecture Notes
-FuzzyCLIPS User Guide

Strategy

This project will incorporate FuzzyCLIPS to provide a framework for the system. The PID control loop will be programmed using standard practices in FuzzyCLIPS with the potential for a C framework if it is necessary. However, this will be different from traditional PID control loops in that the error signal will undergo fuzzification.

If it is deemed feasible, the output signal may also undergo fuzzification.

Outcome/Deliverables:

The goal of the project is to create a functional PID loop controller within the FuzzyCLIPS framework. This PID loop controller will make use of fuzzification of the error signal. A successful project will allow for a stable plant output using fuzzy logic in the FuzzyCLIPS framework.

Development:

This project was meant to explore the use of fuzzy logic as applied to a PID loop controller. The original goal was to have a fully functional PID loop controller that could automatically return a system to a new, stable output given an input perturbance such as a user input or system change. Although it was not defined in the project scope, it was hoped that this could provide a damped response curve using the FuzzyCLIPS ASCII output feature. Unfortunately, many of these features were unable to be implemented due to time and knowledge constraints.

The resultant controller makes use of fuzzification of an input signal to determine what the actual error compared to the existing plant value will be. This then propagates a “preerror” signal through the controller which sets the various controller sections up for performing work. Due to complications associated with FuzzyCLIPS and CLIPS, a correctly looping system that returns to a stable output is not possible within the code without continuously manually re-running the applicable steps. Ultimately, the code is capable of consuming a user input and, by manually applying the correct code using the blocks shown in Figures 1-6 in the correct sequences, move the plant output through iterations that reach the desired user set value.

Results:

Overall, the project can generally be said to be successful. Each major portion of a PID loop controller was approximated using fuzzy controls. The overall plant response was approximated using categorized values. The end result of the plant was that it could raise or lower crisp plant output according to fuzzy input error signals. This is demonstrated by a text output informing the user of a change in the plant output as well as a global variable change indicating the new plant output.

Outcomes:

This project met the defined goals and deliverables as outlined in the proposal. Further development of the fuzzy logic portion of the project could enhance the overall success of the project in making a more efficient controller. The current framework only allows a demonstration of the potential of the system rather than a truly independent system

```
;; PID Fuzzy Controller
;; By Tyler Hardy

;; Deftemplates

;; Global Variables

(defglobal ?*plantOutput* = 70)

;; Input and fuzzification
```

```
(defrule getSetpoint
        (declare (salience 1000))
        =>
        (printout t "Set system temperature: ")

        (bind ?t (read))
        (if (< ?t 65)
                then
                (assert (Setpoint low))
        )

        (if (< ?t 76)
                then
                (if (> ?t 64)
                then
                        (assert (Setpoint medium))
                )
        )

        (if (> ?t 75)
                then
                (assert (Setpoint high))
        )


        (if (> ?t ?*plantOutput*)
                then
                        (assert (preerror low))
        )

        (if (= ?t ?*plantOutput*)
                then
                        (assert (preerror none))
                )
        )

        (if (> ?t ?*plantOutput*)
                then
                        (assert (preerror high))
        )
)
```

Figure 1 – Input Code

```
;; Proportional Error
(defrule pError1
        (declare (salience 50))
        (preerror low)
        =>
        (assert (Perror low))
)

(defrule pError2
        (declare (salience 50))
        (preerror none)
        =>
        (assert (Perror none))
)

(defrule pError3
        (declare (salience 50))
        (preerror high)
        =>
        (assert (Perror high))
)

(defrule pError4
        (declare (salience 50))
        (preerror lowlow)
        =>
        (assert (Perror lowlow))
)

(defrule pError5
        (declare (salience 50))
        (preerror highhigh)
        =>
        (assert (Perror highhigh))
)
```

Figure 2 – Proportional Code

```
;; Integral Error
;; preerror+error

(defrule IError1
        (declare (salience 70))
        (error low)(preerror low)
        =>
        (assert (Ierror low))
)

(defrule IError2
        (declare (salience 70))
        (error none)(preerror low)
```

```
            =>
            (assert (Ierror none))
)

(defrule IError3
            (declare (salience 70))
            (error high)(preerror low)
            =>
            (assert (Ierror high))
)

(defrule IError4
            (declare (salience 70))
            (error low)(preerror none)
            =>
            (assert(Ierror low))
)

(defrule IError5
            (declare (salience 70))
            (error none)(preerror none)
            =>
            (assert(Ierror none))
)

(defrule IError6
            (declare (salience 750))
            (error high)(preerror none)
            =>
            (assert (Ierror high))
)

(defrule IError7
            (declare (salience 70))
            (error low)(preerror high)
            =>
            (assert(Ierror none))
)

(defrule IError8
            (declare (salience 70))
            (error none)(preerror high)
            =>
            (assert (Ierror none))
)

(defrule IError9
            (declare (salience 70))
            (error high)(preerror high)
            =>
            (assert(Ierror high))
```

```
)
```

Figure 3 – Integral Code

```
;; Derivative Error
;; error - preerror

(defrule DError1
	(declare (salience 60))
	(error low)(preerror low)
	=>
	(assert (Derror none))
)

(defrule DError2
	(declare (salience 60))
	(error none)(preerror low)
	=>
	(assert (Derror high))
)

(defrule DError3
	(declare (salience 60))
	(error high)(preerror low)
	=>
	(assert (Derror high))
)

(defrule DError4
	(declare (salience 60))
	(error low)(preerror none)
	=>
	(assert (Derror low))
)

(defrule DError5
	(declare (salience 60))
	(error none)(preerror none)
	=>
	(assert (Derror none))
)

(defrule DError6
	(declare (salience 60))
	(error high)(preerror none)
	=>
	(assert (Derror high))
)
```

```
(defrule DError7
          (declare (salience 60))
          (error low)(preerror high)
          =>
          (assert (Derror low))
)

(defrule DError8
          (declare (salience 60))
          (error none)(preerror high)
          =>
          (assert (Derror low))
)

(defrule DError9
          (declare (salience 60))
          (error high)(preerror high)
          =>
          (assert (Derror none))
)
```

Figure 4 – Derivative Code

```
;; Pre-Summation

(defrule presum1
          (declare (salience -100))
          (Ierror low)(Derror low)
          ?f <- (Ierror low)
          ?g <- (Derror low)
          =>
          (retract ?f)
          (retract ?g)
          (assert (Nerror low))
)

(defrule presum2
          (declare (salience -100))
          (Ierror low)(Derror none)
          ?f <- (Ierror low)
          ?g <- (Derror none)
          =>
          (retract ?f)
          (retract ?g)

          (assert (Nerror low))
)
```

```
(defrule presum3
	(declare (salience -100))
	(Ierror none)(Derror low)
	?f <- (Ierror none)
	?g <- (Derror low)
	=>
	(retract ?f)
	(retract ?g)

	(assert (Nerror low))
)


(defrule presum4
	(declare (salience -100))
	(Ierror none)(Derror none)
	?f <- (Ierror none)
	?g <- (Derror none)
	=>
	(retract ?f)
	(retract ?g)

	(assert (Nerror none))
)

(defrule presum5
	(declare (salience -100))
	(Ierror low)(Derror high)
	?f <- (Ierror low)
	?g <- (Derror high)
	=>
	(retract ?f)
	(retract ?g)

	(assert (Nerror none))
)

(defrule presum6
	(declare (salience -100))
	(Ierror high)(Derror low)
	?f <- (Ierror high)
	?g <- (Derror low)
	=>
	(retract ?f)
	(retract ?g)

	(assert (Nerror none))
)

(defrule presum7
	(declare (salience -100))
	(Ierror high)(Derror high)
	?f <- (Ierror high)
	?g <- (Derror high)
	=>
	(retract ?f)
	(retract ?g)

	(assert (Nerror high))
)

(defrule presum8
	(declare (salience -100))
	(Ierror high)(Derror none)
	?f <- (Ierror high)
	?g <- (Derror none)
	=>
	(retract ?f)
	(retract ?g)

	(assert (Nerror high))
)

(defrule presum9
	(declare (salience -100))
	(Ierror none)(Derror high)

	?f <- (Ierror none)
	?g <- (Derror high)
	=>
	(retract ?f)
	(retract ?g)

	(assert (Nerror high))
)

;; Full Error Summation

;;(defrule sum1
;;	(declare (salience -500))
;;	(Perror lowlow)(Nerror low)
;;	=>
;;	(retract (Perror lowlow)(Nerror low))
;;	(assert (error low))
;;)

;;(defrule sum2
;;	(declare (salience -500))
;;	(Perror lowlow)(Nerror none)
;;	=>
;;	(retract (Perror lowlow)(Nerror none))
;;	(assert (error low))
```

```
;;)

;;(defrule sum3
;;        (declare (salience -500))
;;        (Perror lowlow)(Nerror high)
;;        =>
;;        (retract (Perror lowlow)(Nerror high))
;;        (assert (error none))
;;)

(defrule sum4
        (declare (salience -500))
        (Perror low)(Nerror low)
        ?f <- (Perror low)
        ?g <- (Nerror low)
        =>
        (retract ?f)
        (retract ?g)
        (assert (error low))
)

(defrule sum5
        (declare (salience -500))
        (Perror low)(Nerror none)
        ?f <- (Perror low)
        ?g <- (Nerror none)
        =>
        (retract ?f)
        (retract ?g)
        (assert (error none))
)

(defrule sum6
        (declare (salience -500))
        (Perror low)(Nerror high)

        ?f <- (Perror low)
        ?g <- (Nerror high)
        =>
        (retract ?f)
        (retract ?g)

        (assert (error none))
)

(defrule sum7
        (declare (salience -500))
        (Perror none)(Nerror low)

        ?f <- (Perror none)
        ?g <- (Nerror low)
```

```
            =>
            (retract ?f)
            (retract ?g)

            (assert (error none))
)

(defrule sum8
            (declare (salience -500))
            (Perror none)(Nerror none)

            ?f <- (Perror none)
            ?g <- (Nerror none)
            =>
            (retract ?f)
            (retract ?g)

            (assert (error none))
)

(defrule sum9
            (declare (salience -500))
            (Perror none)(Nerror high)

            ?f <- (Perror low)
            ?g <- (Nerror high)
            =>
            (retract ?f)
            (retract ?g)

            (assert (error none))
)

(defrule sum10
            (declare (salience -500))
            (Perror high)(Nerror low)

            ?f <- (Perror high)
            ?g <- (Nerror low)
            =>
            (retract ?f)
            (retract ?g)

            (assert (error none))
)

(defrule sum11
            (declare (salience -500))
            (Perror high)(Nerror none)

            ?f <- (Perror high)
```

```
        ?g <- (Nerror none)
        =>
        (retract ?f)
        (retract ?g)

        (assert (error none))
)

(defrule sum12
        (declare (salience -500))
        (Perror high)(Nerror high)

        ?f <- (Perror high)
        ?g <- (Nerror high)
        =>
        (retract ?f)
        (retract ?g)

        (assert (error high))
)

;;(defrule sum13
;;      (declare (salience -500))
;;      (Perror highhigh)(Nerror low)

;;      ?f <- (Perror highhigh)
;;      ?g <- (Nerror low)
;;      =>
;;      (retract ?f)
;;      (retract ?g)

;;      (assert (error none))
;;)

;;(defrule sum14
;;      (declare (salience -500))
;;      (Perror highhigh)(Nerror none)

;;      ?f <- (Perror highhigh)
;;      ?g <- (Nerror none)
;;      =>
;;      (retract ?f)
;;      (retract ?g)
;;      (assert (error high))
;;)

;;(defrule sum15
;;      (declare (salience -500))
;;      (Perror highhigh)(Nerror high)

;;      ?f <- (Perror highhigh)
;;          ?g <- (Nerror high)
;;          =>
;;          (retract ?f)
;;          (retract ?g)
;;          (assert (error high))
;;)
```

Figure 5 – Summation Code

```
;; Plant

(defrule plantCooldown
            (declare (salience 1))
            (error low)
            ?f <- (error low)
            =>
;;          (retract ?f)
            (assert (preerror low))
            (bind ?*plantOutput* (- ?*plantOutput* 1))
            (printout t "Lowering plant output" crlf)
)

(defrule plantSS
            (declare (salience 1))
            (error none)
            ?f <- (error none)
            =>
;;          (retract ?f)
            (assert (preerror none))
            (printout t "Maintaining plant output" crlf)
)

(defrule plantHeatup
            (declare (salience 1))
            (error high)
            ?f <- (error high)
            =>
;;          (retract ?f)
            (assert (preerror high))
            (bind ?*plantOutput* (+ ?*plantOutput* 1))
            (printout t "Increasing plant output" crlf)
)
```

Figure 6 – Plant and Output Code

This project could benefit from improved flow in the execution sequence. Currently the project relies on manual entry of each rule sequence to guarantee execution. In order to demonstrate a PID loop controller most accurately, all relevant rules should fire. Otherwise, the loop controller will work as either a P, PD, PI, or not accurately work as a controller.

Another major area of improvement would be using more fuzzy logic controls within the project framework. Currently the project implements fuzzy logic concepts using a syntax that is executable within FuzzyCLIPS. This could be further adapted to include probabilistic error correction.

The final area of improvement for this project would be in actual state modeling of the output of a hypothetical PID fuzzy controller using the FuzzyCLIPS ASCII graphing features. This would require fully elaborating the concept of a PID controller using calculus as well as adapting the equations for fuzzy math.