Hands-on Lab

Lego NXT Domabot – Wall-following PID

An ultrasonic sensor (US) is mounted on the Domabot's portside (left). The US measures the distance of a wall on the Domabot's portside. A PID controller is used to regulate a desired wall-to-portside distance.

Concept 1 – Program Structure:

Step 1: Create a new file called **wfPid1_0a.nxc**

```
// FILE: wfPid1_0a.nxc - Works!
// DATE: 10/27/22 12:22
// AUTH: P.Oh
// DESC: Domabot US sensor (Port 4) senses a wall left of it.  Domabot always
//       turns left (CCW yaw) as it moves along wall. PID or bang-bang (i.e.
//       when PID gains all are zero) is used to calculate turning speed
// VERS: 1.0a:  ME 425 release
// REFS: wfbb0_1a3.nxc; x^2File1.0.nxc; wf2Us0_3a1.nxc

// Global variables ---------------------------------------------------------
  int xWall;                // wall distance [cm]
  int dWall;                // desired wall distance [cm]
  float wKp, wKi, wKd;      // wall PID gains;
  float wE, wEDot, wEInt;   // wall error and its derivative and integral
  float wEPrev;             // wall error previous value
  float wCorr;              // wall related motor power correction [0, 100]
  int speedA, speedC;       // motor speed for A (right) and C (left)
  int speedBase;            // motor base speed
  bool orangeButtonPushed, rightButtonPushed, leftButtonPushed; // NXT buttons

task main() {

  // Variable initializations -----------------------------------------------
  xWall = 0;                // initialize wall distance to 0
  dWall = 10;               // Desired distance from wall [cm]
  wKp = 0.0;                // Wall P gain e.g. (PID)=[1.5, 0.005, 30.0]
  wKi = 0.0;                // Wall I gain
  wKd = 0.0;                // Wall D gain
  wE = wEDot = wEInt = 0.0; // initialize wall error-related values to 0
  wEPrev = 0.0;             // initialize previous wall error to 0
  speedBase = 30;           // Domabot base motor speed at 50% i.e. mid-point

  // Algorithm begins -------------------------------------------------------

  TextOut(0, LCD_LINE2, "-> BTN to proceed" );
  SetSensorLowspeed(IN_4); // Wall on Left US (Port 4)

  do {
    rightButtonPushed = ButtonPressed(BTNRIGHT, FALSE);
    xWall = SensorUS(IN_4); // for wall detection (on left, Port 4)
    TextOut(0, LCD_LINE3, FormatNum("Wall = %3d cm" , xWall));
  } while(!rightButtonPushed);

  ClearLine(LCD_LINE2);
  TextOut(0, LCD_LINE2, "<- BTN to QUIT" );
```

```
  do { // continue wall following until left button pushed
    leftButtonPushed = ButtonPressed(BTNLEFT, FALSE);
    xWall= SensorUS(IN_4);      // left US (Port 4)
    TextOut(0, LCD_LINE3, FormatNum("Wall = %3d cm" , xWall));
    // (1) Calculate wall-following PID gains
```

Add PID control effort here by calculating error, derivative of error, and integral of error

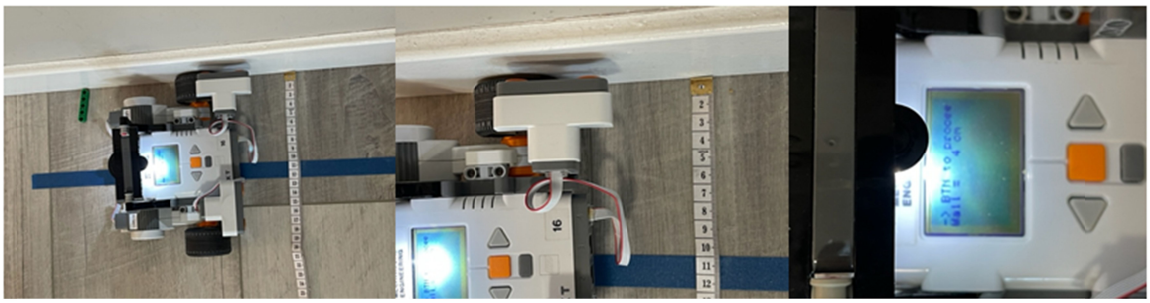
```
    // (1A) Check for motor staturation i.e. resulting wCorr forces
    // motor getting > 2*speedBase (if speedBase > 50, this means > 100)
    if(wCorr > 0 && wCorr > speedBase) {
       wCorr = speedBase;       // saturated so set correction to speedBase
                                // So Motor A speed max will be 2*speedBase
    };
    if(wCorr < 0 && wCorr < -speedBase) {
        wCorr = -speedBase;     // saturated so set correction to -speedBase
                                // So Motor C speed min will be 0
    };
    // (1B) If PID gains all zero, then wCorr = 0 so do bang-bang
    if(wCorr == 0 && xWall < dWall) {
          wCorr = -speedBase;  // Move away from wall: C = basespeed, A = 0
    };
    if(wCorr == 0 && xWall >= dWall) {
          wCorr = speedBase;   // Move towards wall: A = basespeed, C = 0
    };

    // (2) Command motors
    speedA = speedBase + wCorr;
    speedC = speedBase - wCorr;
    OnFwd(OUT_C, speedC);
    OnFwd(OUT_A, speedA);
    // (3) update wall errors for next derivative calculation
    wEPrev = wE;
  } while( (!leftButtonPushed)  ); // end do-loop

   // (4) User pushed <-- (Left) Button, so exit gracefully
   Off(OUT_AC);
   PlaySound(SOUND_DOUBLE_BEEP);
   Wait(5000);
   StopAllTasks();

} // end main

} // end of main
```

Step 2: Using the variables declared in **wfPid1_0a.nxc** one sees that the current wall error `wE` is the difference between the measured wall distance `xWall` and desired wall distance `dWall`. That is `wE = xWall - dWall`. The derivative of the wall error `wEDot` is just the difference between the current wall error `wE` and previous wall error `prevWPrev`. Highlighted in yellow one sees that before the loop returns for the next iteration, the previous wall error `wEPrev` is set equal to the current wall error `wE`. Lastly, the sum (i.e. integral) of the wall error `wEInt` is the sum of current wall error `wE` and previous sum of the wall errors `wEInt`. With this knowledge one can add PID correction as

```
wCorr = wKp * wE + wKi * wEInt + wKd * wEDot;
```

Step 3: Contrast bang-bang and P-only control performance. For bang-bang set `wKp = wKi = wKd = 0.0`. For P-only control, set `wKp = 1.5; wKi = 0.0;` and `wKd = 0.0`. Set the Domabot 4 to 5 cm from the wall (see photos below) and observe its performance trying to maintain a wall-to-portside distance of 10 cm (marked by blue painters tape on the floor).

Blue tape is 10 cm from wall baseboard. Ultrasonic sensor shows portside sensor at 10 cm



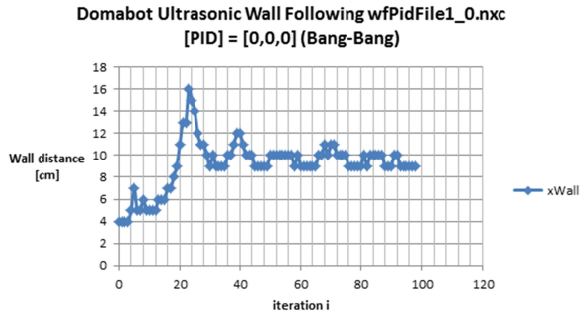
Domabot portside tire touches the baseboard. Ultrasonic sensor shows this is 4 cm

Concept 2 – Data acquisition of wall-following performance

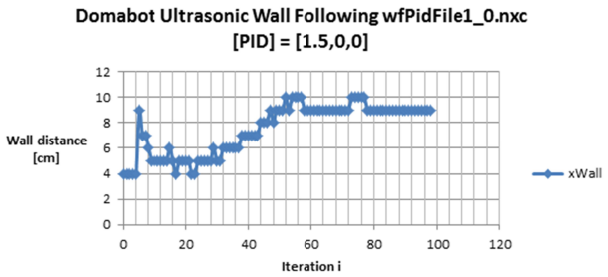
Step 1: Create a new file called **`wfPidFile1_0a.nxc`**

Recalling `w^2File1_0.nxc`, add file saving functions to `wfPidFile1_0a.nxc`.

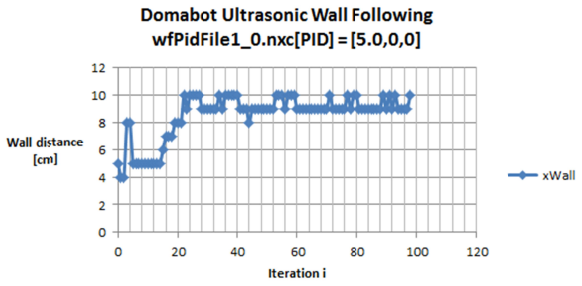
Step 2: Re-run your programs to contrast bang-bang and P-only control. Confirm that your Excel plots look similar to the following



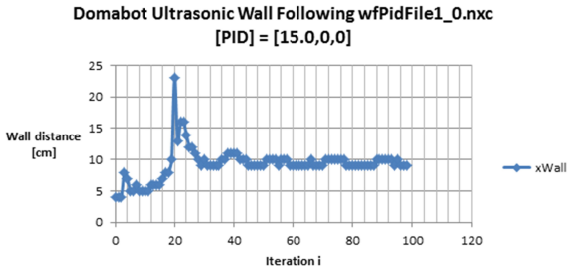
YouTube: https://youtu.be/DlVmcWBeccw



YouTube: https://youtu.be/edauCdpyZMA



YouTube: https://youtu.be/edauCdpyZMA



YouTube: https://youtu.be/YWy7kzwAUnM

Figure A: Excel plots of bang-bang and P-only control