Hands-on Lab

Lego NXT Domabot – Obstacle Avoidance (OA) PID

An ultrasonic sensor (US) is mounted on the Domabot's bow (front). This enables the Domabot to sense obstacles in front of it. A PID controller is used to turn right (yaw CW) around the obstacle.

Concept 1 – Program Structure:

Step 1: Create a new file called **oaPid1_0a.nxc**

```
// FILE: oaPid1_0a.nxc - Works!
// DATE: 10/30/22 11:40
// AUTH: P.Oh
// DESC: Domabot obstacle avoidance with ultrasonic sensor (US)
//       Obstacle in front US (Port 1) with PID
// VERS: 1.a: ME 425 release
// REFS: wfbb0_1a3.nxc; x^2File1.0.nxc; wf2Us0_3a1.nxc; oaPid0_1b.nxc

// Global variables ---------------------------------------------------------
  int xObst;                // obstacle distance [cm]
  int dObst;                // desired obstacle distance [cm]
  float oKp, oKi, oKd;      // obstacle PID gains;
  float oCorr;              // obstacle related motor power correction [0, 100]
  float oE, oEDot, oEInt;   // obstacle error and its derivative and integral
  float oEPrev;             // obstacle error previous value
  int speedA, speedC;       // motor speed for A (right) and C (left)
  int speedBase;            // motor base speed
  bool rightTurnBegin;      // Domabot began right (CW) yaw to avoid obstacle
  bool straightLineBegin;   // Domabot began wall-following
  bool orangeButtonPushed, rightButtonPushed, leftButtonPushed; // NXT buttons

task main() {

  // Variable initializations -----------------------------------------------
  xObst = 0;                // initialize obstacle distance to 0
  dObst = 30;               // Desired distance to obstacle [cm]
  oKp = 20.0;               // Obstacle P gain e.g. 20
  oKi = 0.01;               // Obstacle I gain e.g. 0.01
  oKd = 0.5;                // Obstacle D gain e.g. 0.5
  oE = oEDot = oEInt = 0.0; // initialize obstacle error-related values to 0
  oEPrev = 0.0;             // initialize previous wall and obstacle errors to 0
  speedBase = 50;           // Domabot base motor speed at 50% i.e. mid-point
  rightTurnBegin = straightLineBegin = FALSE; // initially FALSE

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

  TextOut(0, LCD_LINE2, "-> BTN to proceed" );
  SetSensorLowspeed(IN_1); // Obstacle in Front US (Port 1)

  do {
    rightButtonPushed = ButtonPressed(BTNRIGHT, FALSE);
    xObst = SensorUS(IN_1); // for object detection (front, Port 1)
    TextOut(0, LCD_LINE4, FormatNum("Obst = %3d cm" , xObst));
  } while(!rightButtonPushed);

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

© Copyright Paul Oh

```
  do { // continue obstacle avoiding (OA) until left button pushed
    leftButtonPushed = ButtonPressed(BTNLEFT, FALSE);
    xObst = SensorUS(IN_1);     // sense obstacle distance
    TextOut(0, LCD_LINE4, FormatNum("Obst = %3d cm" , xObst));

    // (0) Just start moving forward
    OnFwd(OUT_AC, speedBase);

    // (1) Check for obstacle and calculate PID control for sharp CW yaw
    while( xObst <= dObst ) { // Domabot detected obstacle.  Begin right turn
        PlayTone(TONE_A7, 1);
        // (1A) If first time avoiding obstacle...
        if(rightTurnBegin == FALSE) {
           // ...then clear gains to avoid kick and windup
           oEDot = oEInt = oEPrev = 0.0;
        }; // end if(rightTurnRight == FALSE)
        xObst = SensorUS(IN_1); // sense obstacle distance
        // (1B) Calculate PID control effort to avoid obstacle
```

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

```
        if(oCorr > 0 && oCorr > speedBase) oCorr = speedBase;   // saturated
        if(oCorr < 0 && oCorr < -speedBase) oCorr = -speedBase; // saturated
        if(oCorr == 0) oCorr = 0; // just bang-bang
        // (1C) Right turn is CW yaw so make motor speed C > motor speed A
        speedC = speedBase - oCorr; // NB: oCorr < 0 and saturation sets oCorr
        speedA = oCorr - speedBase; // to -speedBase, C = 2*speedBase and A = 0
        OnFwd(OUT_C, speedC);
        OnFwd(OUT_A, speedA);
        // (1D) Domabot has turned (and turning) right i.e. CW yaw
        rightTurnBegin = TRUE;      // TRUE, right turn began
        straightLineBegin = FALSE;  // FALSE, Domabot no longer going straight
        oEPrev = oE; // update obstacle errors for next derivative calculation
    }; // end while xObst <= dObst

    // (2) No obstacle to avoid so go straight
    straightLineBegin = TRUE;  // TRUE, Domabot going straight now
    rightTurnBegin = FALSE;    // FALSE, Domabot no longer turning right

  } 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
```

Step 2: The aqua highlighted lines introduce Boolean variables. These serve as flags. One observes that they are initialized FALSE. When the Domabot detects and obstacle and begins turning right for the first time, the derivative, integral, and previous error values (`oEDot, oEInt, oEPrev`)are reset to zero. This is to avoid derivative kick and integral windup. In other words, one wants to reset the corrective control effort. After the turn starts, `rightTurnBegin` is set to TRUE and PID is used to turn the turning speed.

Step 3: Using the variables declared in **`oaPid1_0a.nxc`** one sees that the current obstacle error `oE` is the difference between the measured obstacle distance `oWall` and desired obstacle distance `oWall`. That is `oE = oWall – oWall`. The derivative of the obstacle error `oEDot` is just the difference between the current obstacle error `wE` and previous obstacle error `prevOPrev`. Highlighted in yellow one sees that before the loop returns for the next iteration, the previous obstacle error `oEPrev` is set equal to the current wall error `oE`. Lastly, the sum (i.e.

© Copyright Paul Oh

integral) of the obstacle error `oEInt` is the sum of current obstacle error `oE` and previous sum of the obstacle errors `oEInt`. With this knowledge one can add PID correction as

```
oCorr = oKp * oE + oKi * oEInt + oKd * oEDot;
```

Step 3: There are 3 things one can tune to change performance: (1) the bow-to-obstacle distance `oObst`; (2) the base speed, `speedBase`; and (3) the PID gains `oKp`, `oKi`, and `oKd`.

Construct a walled arena like the photo below. Qualitatively describe the performance as you change the 3 tuning parameters.

Walled arena approximately 36 inches by 48 inches

Trial	`speedBase`	`oObst`	[`oKp`, `oKi`, `oKd`]	Observations
1	30	15	[20, 0.01, 0.5]	
2	30	30	[20, 0.01, 0.5]	
3	50	30	[20, 0.01, 0.5]	
4	50	30	[0, 0, 0]	
5	30	30	[0, 0, 0]	
6	50	15	[20, 0.01, 0.5]	