Hands-on Lab

Visual Servoing the XL-320 Lego 2-link Planar Manipulator

The previous distributed computing lab demonstrated that one can split processes among a PC and a pair of Master and Slave NXT Bricks. This is especially useful in visual servoing. Here the PC performs the computation-heavy image processing and the NXT just performs less intensive computations like inverse kinematics and motor servoing.

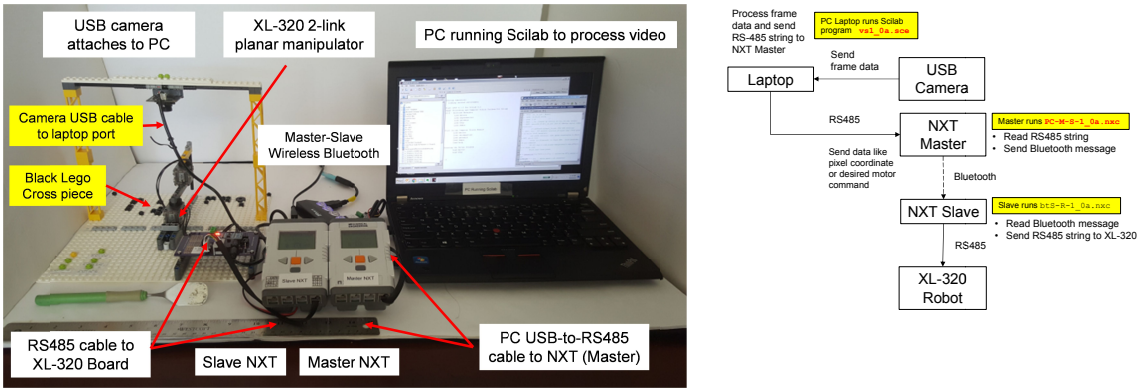

Figure A: Almost identical to the figure in the Distributed Computing lab, one sees the camera's USB cable and black Lego Cross piece (yellow boxes) added (left). The flow diagram (right) now runs the image processing program vs1_0a.sce which incorporates serial communications. Demo video: https://youtu.be/gUJ1-CzoqHk .

Concept: Open-loop visual-servoing `vs1_0a.sce`

Figure A (right) is the flow diagram. One notices that there is no feedback loop from the XL-320 robot to the laptop. This is for simplicity; the PC shall detect the location of a template in a single frame grabbed by camera. Then the mapping from image space to robot task space is performed. The location passes along NXT Bricks serially and via Bluetooth to ultimately command the XL-320 2-link planar manipulator.

As such, the Master NXT runs the same **`PC-M-S-1_0a.nxc`** (renamed **`PC-M-S-IK-1_0a.nxc`**) code. The Slave modifies **`btS-R-1_0a.nxc`** to incorporate the XL-320 Lego 2-link planar manipulator's inverse kinematics (which can be called **`btS-R-IK-1_0a.nxc`**). As such, the only new program needed is the Scilab one.

Step 1: For the PC, write a Scilab (version 6.1.0) program that adds SSD tracking

The listing is given in **Figure 1A**. It looks long, but most of the code is the same as the one in the Distributed Computing lab (`serialPc-M-1_0b.sce`). A quick skim recalls the serial port setup and formulation of strings to serially send to the Master NXT. Additionally, one recognizes sections of code from the video processing lab (`scilabHelloVision1_0a.sce` and `scilabTracking1_0a.sce`).

Additions begin with definitions of parameters like the baseplate's width and height, lens focal lengths and lens-to-target distance.

```
// FILE: vs1_0a.sce - Works!
// DATE: 04/24/20 09:54
// AUTH: P.Oh
// VERS: 1_0a: cleaned up version of vs0_1d1.sce (works)
// REFS: serialPc-M-IK-1_0a.sce (for PC->Master serial communications)
//       scilabTrackingLego0_1a.sce (for SSD tracking)
// DESC: Goal: SSD detects location of black Lego Cross piece on white 32x32 baseplate
//       and sends serially, task space coordinates to Master NXT, which then
//       sends via Bluetooth, to Slave NXT.  Slave performs IK and commands
//       XL-320 Lego 2-link planar manipulator's end-effector to hover over
//       Black Lego Cross piece

h = openserial(10,"4800,n,8,1"); // initialize PC's serial port
strHeader = " @"; // white space + at character
stringRoger = "ROGER";
stringRogerFound = 1; // not TRUE

// Definitions
W = 14.5*8;     // = 116 mm = Lego 32x32 baseplate width from image file shows 14.5 studs
H = 15.0*8;     // = 120 mm = Lego 32x32 baseplate height from image file shows 15 studs
u = 0;          // row location wrt image frame [pix]
v = 0;          // col location wrt image frame [pix]
x = 0;          // row location wrt robot frame [mm]
y = 0;          // col location wrt robot frame [mm]
fRow = 278.65;  // [pix] focal length along image rows
fCol = 277.89;  // [pix] focal length along image cols
z = 103;        // [mm] target-to-lens distance

// (A) Initialize Scilab Computer Vision Module; Get ID of webcam; Setup graphic window
scicv_Init();
// Usually 0: computer's build-in webcam; 1: USB webcam.  If 1 doesn't work try 2
videoCapture = new_VideoCapture(2);
f =scf(0); // set current graphic figure
[ret, frame] = VideoCapture_read(videoCapture); // grab and return a frame

subplot(1,3,1); // Set up 1 row and 2 columns of sub-plots.  Draw in Plot 1
matplot(frame);
disp("Size:");
disp(size(frame));
disp("Number of cols:");
disp(Mat_cols_get(frame));
disp("Number of rows:");
disp(Mat_rows_get(frame));

counterFlag = 0; // just want to save one frame to file

// (A) Endless loop that grabs frame, displays it, and repeats
while is_handle_valid(f)
    [ret, frame] = VideoCapture_read(videoCapture); // grab and return a frame
    // (A-1) Video seen by camera (left); grey-scale(middle); threshold (right)
    if is_handle_valid(f) then
        // ret is TRUE, so display frame
        subplot(1,3,1); // Set up 1 row and 2 columns of sub-plots.  Draw in Plot 1
        matplot(frame);
        greyFrame = cvtColor(frame, CV_BGR2GRAY);
        subplot(1,3,2);
        matplot(greyFrame);
        thresholdValue = 30; // 0 (whiter stuff becomes white) and 255 (blacker stuff becomes
black)
        [thresh, thresholdedFrame]  = threshold(greyFrame, thresholdValue, 255,
THRESH_BINARY);
        subplot(1,3,3);
        matplot(thresholdedFrame);
```

Figure 1A: Scilab program **vs1_0a.sce**

```
        if counterFlag == 10 then // Grab (10th) )frame after video starts/settles
            // (B)Perform SSD
            // (B-1) Grab a single image.  NB: define path where to save image files
            imwrite(fullfile("H:\00courses\me7XX\XX-visualServoing",
"thresholdedFrame.png"), thresholdedFrame);
            imwrite(fullfile("H:\00courses\me7XX\XX-visualServoing", "greyFrame.png"),
greyFrame);

            // (B-2) Perform SSD and find target center location in pixels
            img = imread("H:\00courses\me7XX\XX-visualServoing\thresholdedFrame.png");
            img_template = imread("H:\00courses\me7XX\XX-visualServoing\template.png");
            img_result = matchTemplate(img, img_template, CV_TM_SQDIFF); // 0 = match
            [min_value, max_value, min_value_loc, max_value_loc] = minMaxLoc(img_result)
            disp("min_value =");
            disp(min_value);
            disp("location in image:")
            disp(min_value_loc);
            u = min_value_loc(2); // [pix]
            v = min_value_loc(1); // [pix]

            // (B-3) Calculate target center.  Recall target template is 66 rows and 66
cols
            uCenter = u + (66/2); // [pix]
            vCenter = v + (66/2); // [pix]
            // (B-4) Convert pix to mm
            uCenterMM = (z * uCenter)/fRow; // [mm]
            vCenterMM = (z * vCenter)/fCol; // [mm]
            disp("uCenter [mm] = ");
            disp(uCenterMM);
            disp("vCenter [mm] = ");
            disp(vCenterMM);
            // (B-5) Convert image space 0_I to robot task space O_R) coordinates
            if vCenterMM <= W then
               // target in +X and +Y quadrant
               x = H - uCenterMM; // [mm]
               y = W - vCenterMM' // [mm]
            else
               // target in +X and -Y quadrant
               x = H - uCenterMM; // [mm]
               // --- y = vCenterMM - W; // [mm]
               y = -(vCenterMM - W); // [mm]
            end;
            positionX = round(x);
            positionY = round(y);
            disp("x [mm] = ");
            disp(positionX);
            disp("y [mm] = ");
            disp(positionY);
            // (B-6) Convert positions into string to send serially to Master NXT
            strPositionX = string(positionX);
            strPositionY = string(positionY);
            strI = strcat([strHeader, strPositionX, ", ", strPositionY]);
            disp(strI);
```

Figure 1A continued: `vs1_0a.sce`

```
            // (C) Send coordinates PC->Master
            // (C-1) serially transmit target's center to Master NXT
            writeserial(h, strI);
            buf = readserial(h);
            // (C-2)Check if Master ready to receive next string
            stringRogerFound = strcmp(stringRoger, buf); // 0: means identical strings
            while (stringRogerFound ~=0) // then NXT -> PC string not ROGER, so wait
                buf = readserial(h);
                stringRogerFound = strcmp(stringRoger, buf);
                sleep(200); // min about 50 ms before reading serial port again
            end; // exit reading serial port when ROGER received
            disp(buf);
            sleep(5000); // just slows down loop so user can see what's happening
            disp("All done!");
            // (C-3) gracefully terminate program
            closeserial(h);
            disp("Closed serial port");
            // sleep(500); // Not needed but uncomment if want time to read console
            delete("all"); // kill all frames
            delete(f); // kill the set graphic figure
            disp("Closed graphics windows");
            // sleep(500); // Not needed but uncomment if want time to read console
            delete_VideoCapture(videoCapture);
            disp("Closed Video Capture");
            // sleep(500);  // Not needed but uncomment if want time to read console
            abort; // This just exits of the program without killing Scilab
        end
        counterFlag = counterFlag + 1;
    end // end if
end // end while
// close gracefully if user quits process manually
closeserial(h);
delete("all"); // kill all frames
delete(f); // kill the set graphic figure
delete_VideoCapture(videoCapture);
```

Figure 1A continued: `vs1_0a.sce`

After a call to `scicv_Init` the code enters Step (A) with an endless `while-loop` that displays the camera's raw, greyscale and thresholded views. Step (B) grabs an image frame (10th in this example) and performs SSD tracking using a pre-defined `template.png` file. Steps (B-2) thru (B-6) map the image space locations to robot task space ones and creates the necessary string.

Step (C) serially sends this string to the Master NXT. Once completed in Step (C-3), the program exits (by calling `abort`) but first closes the serial port, all display frames, the graphic figure, and video connection.

Closing Remarks

Project 2 asks one to demonstrate open-loop visual servoing. As such, the details for writing the NXC programs are left to the user. As mentioned in the beginning, these programs are very similar to those introduced in earlier labs.

One could improve `vs1_0a.sce` with more advanced Scilab programming like graphical user interfaces (GUIs) and timing controls. SSD is OK, but more advanced similarity measures could be implemented. Moreover, unwarping techniques can be implemented to account for skewed images. These are beyond the scope of this course, but are the next steps for improving visual-servoing performance and adding closed-loop feedback.