# Comp3352 Computer Vision: Object Detection

**C.B.Price January 2021**

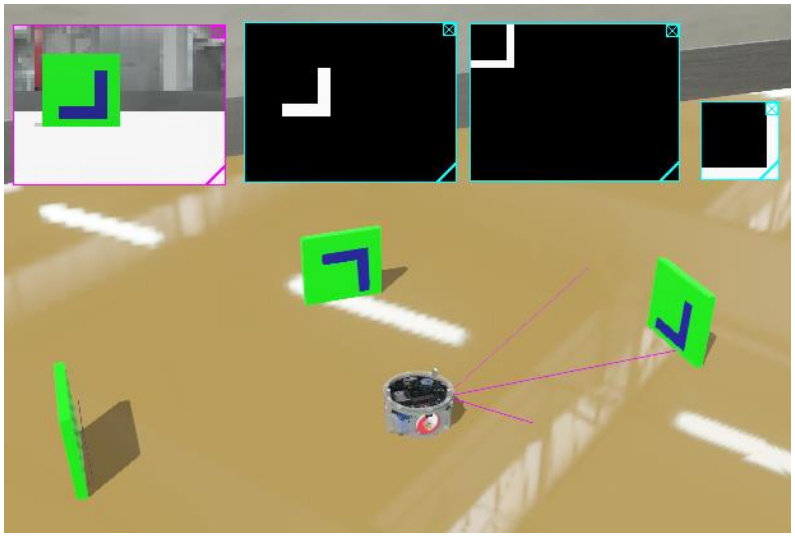
| | |
|---|---|
| **Purpose** | To use computer vision to identify different objects. |
| **Files Required** | Webots project folders on website. Use the world **CBP_3352_IProc_2.wbt** Controller is **CBP_3352_IProc_2.c** |
| **ILO Contribution** | 1 |
| **Send to Me** | If you are working online, please send movie-clip of your solution. |
| **Homework** | |

## Activities

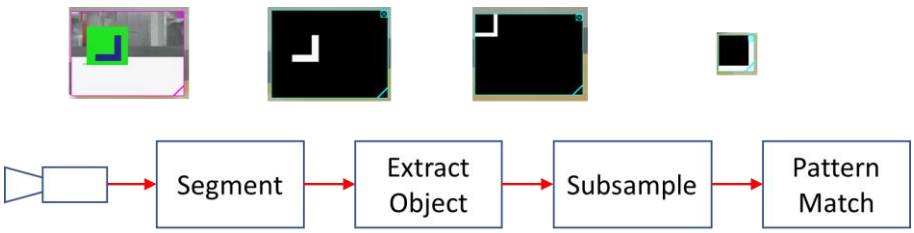
---

**The Task**

Here's the ePuck in the world with three objects each containing an 'L-shape' but with different rotations. The aim of this task is to write some image processing code so the controller will identify which L-shape variant it is looking at. In other words the robot vision system can recognize different objects. The screens on the top show the camera image (leftmost) then various stages in the image processing.



The stages in the image processing sequence are shown below together with the image at the output of each stage.




(i) The camera image is 'segmented' by looking for blue pixels producing a binary image with black or white pixels. (ii) This binary image is scanned for white pixels and so the object is extracted and written into another binary image. (iii) This third image is subsampled to a fixed size of 7x7. (iv) The subsampled image is then compared with the set of 7x7 L-shapes, it outputs a score for each shape.

---

## 1 Building up the code - Segmentation

The code template provided simply shows the camera image when the controller runs. Here you will add the image processing stages and display their results.

Make sure you understand the function of each line you add. This includes the function *arguments* and the function *return* values. It may be a good idea to write these down or talk with a mate.

(a) Add this line of code to create the segmented image – following comment (1) in the code.

**nr_pels = imageSegment(imageCamera,imageSeg,widthCamera,heightCamera);**

and run the code making sure segmentation works.

(b) Now make the segmentation function segment the green pixels: Open up **CBP_IProc_1.c** and change the statement selecting the color range. Get a suitable range from a color wheel (Google 'color wheel degrees'. Make sure this works.

(c) To understand the need to have saturation **sat** in the detection condition, temporarily remove **&& sat > 0.7** from the condition. Look at the resulting mess.

Restore the code to its original state, segmenting on blue with **sat** in the test.

---

## 2 Building up the code – Object Extraction

(a) Add this line of code – following comment (2) in the code

```
nr_pels = detectObjectBounds(imageSeg,imageObj,widthCamera,heightCamera,
                &widthImageObj, &heightImageObj, &tlr,&tlc,&brr,&brc);
```

Note that variables preceded by **&** are outputs from the function call. The variables tlr, tlc, etc refer to the top-left row (tlr), top-left column (tlc), bottom-right row (brr) and bottom-right column (brc) of the rectangle containing the object. This is its 'bounding box'.

(b) In the **#define** up top set DEBUG to true so you can see these values. Look at these and convince yourself they make sense. How? Well the program outputs the height and width of the camera image when it first starts. When complete, set DEBUG to false.

(c) Now we use these values tlr, tlc, etc to extract the object from the segmented image and write it into a new image, **imageObj**. Add the following line – following comment (3) in the code

**nr_pels = writeObjectToImage(imageSeg,imageObj,widthCamera,tlr,tlc,brr,brc);**

You should see the object appears located at the top left of **imageObj**

(d) If you are a coder, and you have inclination, and have time, look at the two functions used here in the source file **CBP_IProc_2.c**

## 3 Building up the code – Subsampling

(a) Add the following code to create the subsampled image **imageSubs** by adding this line after comment (4)

**nr_pels = createSubSampleImage(imageObj,imageSubs,widthImageObj,heightImageObj);**

Check the subsampling works.

(b) Optional (requires revising the code in several places). The above function always subsamples to 7x7. It would be better to pass the subsampled image desired width and height to this function from the controller code. You will need to modify this function and also the controller code (the desired width and height are specified here). You will also need to modify the **Display "display3"** node in the Scene Tree (ePuck – turret slot)

---

## 4 The Pattern Matching Code

(a) Find the call to **imageImageOverlap(…)** in the keyboard handler code, and the function body in **CBP_IProc_3.c**

(b) Run the world, stop the ePuck at each object, and press left-arrow to calculate the overlap with the L-Shape variant **Γ**

---

## 5 Completing the Pattern-Match Code

The code template provided only compares one pattern image **pattern_image** with the subsampled image. This image is created from the array **pattern_1** by a call to **convertArrayToImage(…)** just before the while loop. As you know, the matching is done in a call to **imageImageOverlap(…)** in the keyboard handler code. The actual pattern array is defined and declared in **CBP_IProc3.h**

Your task is to code the remaining three L-Shapes, and to extend/modify the keyboard handler to test the match against all four L-Shapes, or to automate this, so the program will output the best match.

---