Intelligent Robotics
Assignment 1 – iCubSim Robotics Project

Aim

The aim of this report was to further investigate visual attention and close the interaction loop between the robot-simulation (its point of attention) and a visual input. This was carried out by implementing part of a multi-scale saliency model.

The iCub was placed in front of a screen which showed a bi-coloured jagged splodge image which moved around the screen. The software had to read out this image from the iCub camera and apply several filters to it. This then allowed circle detection to be applied allowing the iCub to follow the image around the screen.

This report captures project implementation steps, the issues encountered during the process and, the steps taken to overcome the issues. It is divided into a number of stages, each with one or more steps that were followed in sequentially.

Setup

1. Set up server: Start the yarpserver using the "**yarpserver**" command.
2. Start iCub simulator: Run the "**icub_Sim**" command from the location of the iCub_parts _activation.ini .
3. Open a yarpview window: Open using the **yarpview –name /view/left** command
4. Generate & project image on to screen: This is a two-part command that involves:
   i. The test grabber: **yarpdev --device test_grabber --name /test/video --mode ball**
   ii. Connect the video to the screen: **yarp connect /test/video /icubSim/texture/screen**
5. Enable iCub head movement: Make sure that ikinGaze is running in order to send commands to it and move the icubs head:
   "C:\Program Files (x86)\robotology\icub-1.1.16\bin\iKinGazeCtrl" --robot icubSim

Method

The first step was to read the image from the iCub camera video stream into the program. This was achieved by connecting a self-defined port to the camera port (in this case the left camera was chosen). It was then possible to read from the self-defined port and return the image as an ImageOf<PixelRgb> format.

The next step was to get the simulator to look at the screen while a video was shown to it. By default, the iCub looks straight ahead at the screen *(Figure 1).* If the screen isn't visible it may be due to it being set as 'off' in the iCub_parts *activation.ini* . Note that the video should already be visible to the iCub due to the test_grabber commands used in the setup.

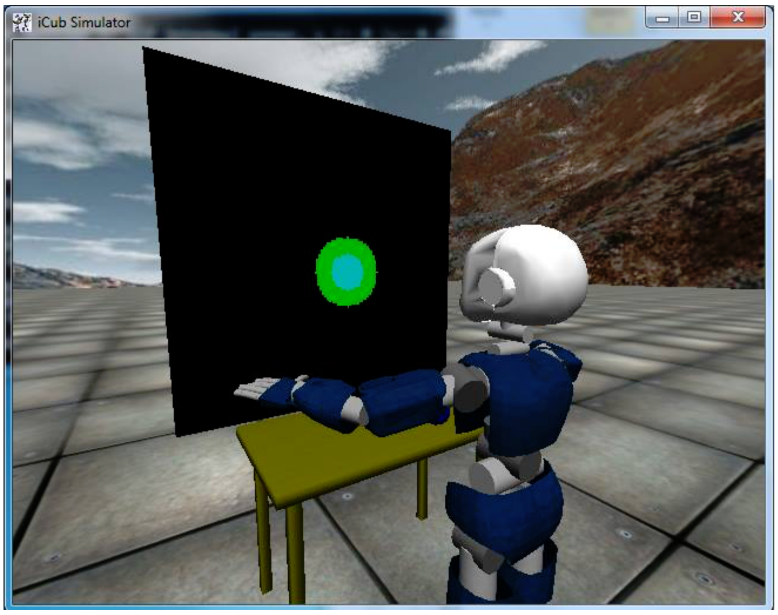


Figure 1: icub looking at the projected video

Image Filtering

Several linear filters were then applied to the image read in by the camera. This was achieved by using openCV. In order to apply these filters, the image needed to be converted from a yarp image to openCV Mat form. It was then possible to show this image in an external window as openCV form *(Figure 2).*

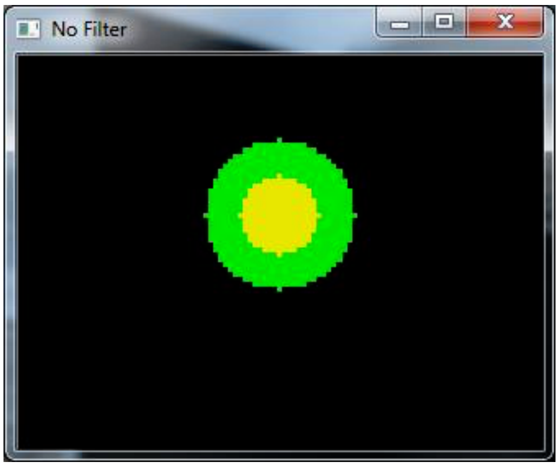


Figure 2. Image from screen with no filter

The aim of the linear filtering was to get the image as clear as possible. First off, the image was converted to grey scale and then two additional blur filters were applied.

By converting the image to grey scale the image was simplified as the image then appeared as two circle of similar colours *(Figure 3)* compared to the contrasting green and yellow of the non filtered image.

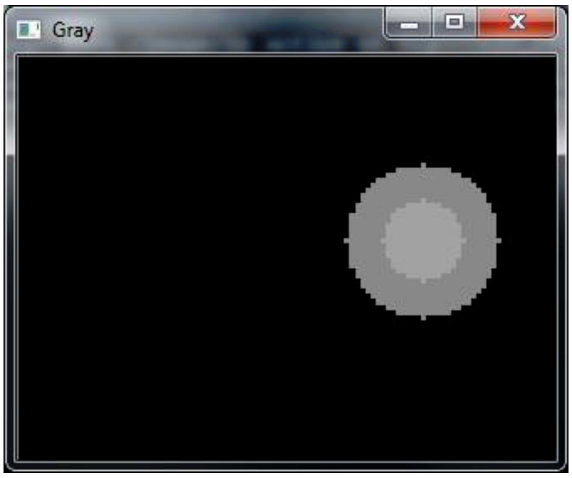


Figure 3: Image in grey scale

The blur filter allowed the edges of the circle to be smoothed *(Figure 4).* This helped when applying the circle detection in a later stage.

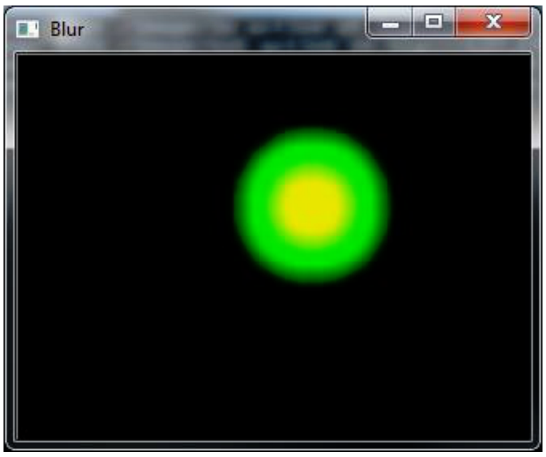


Figure 4: Image with blur applied

By adding the Gaussian blur as well this allowed the image to be smoothed out further into a smooth perfect circle *(Figure 5)*.

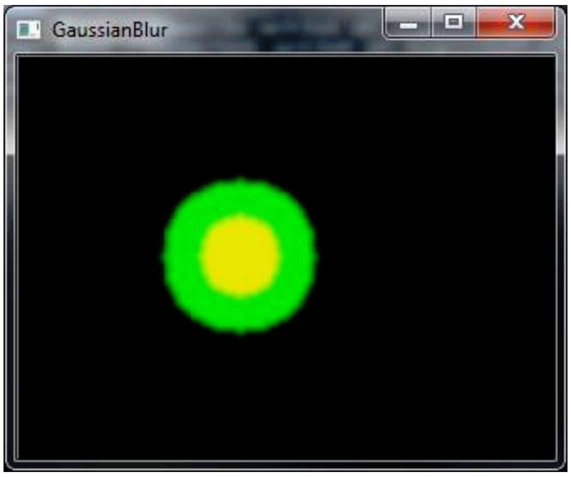


Figure 5: Image with Gaussian blur applied

The resulting image with all the filters produced a nice smooth, single coloured grey circle *(Figure 6)*.

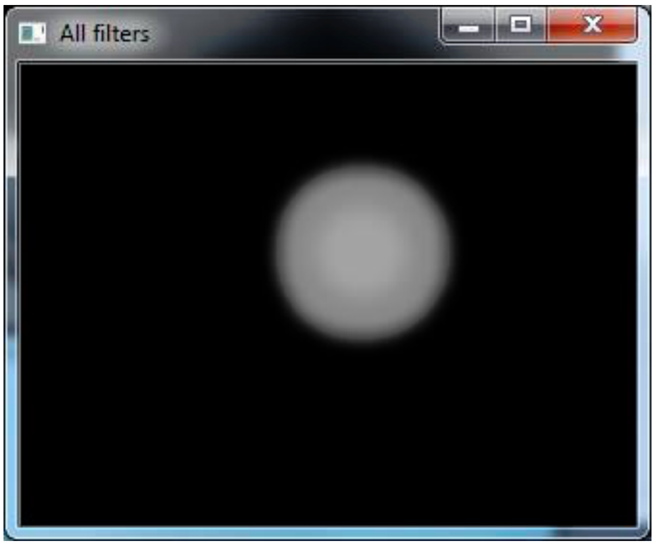
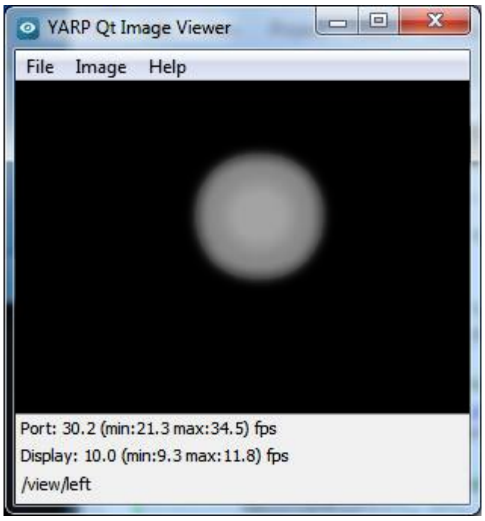


Figure 6: All filters

This image was then converted back into a yarp image so it could be shown in yarpview *(Figure 7)*.



Port: 30.2 (min:21.3 max:34.5) fps
Display: 10.0 (min:9.3 max:11.8) fps
/view/left

Figure 7: Filtered image in yarpview

Circle Detection

Circle detection was then applied on the filtered image using Hough Circle Transform *(Figure 8)*. Three separate loops were use to find the best values for: minimum distance between detected centres, upper threshold for the internal Canny edge detector and threshold for centre detection. By altering these values, it was possible to find the ideal circle detection for the filtered image. Once the circle was obtained, the centre of the circle could then be passed back as a point with an x and y value. This circle edge and centre can then be shown on top of the original image

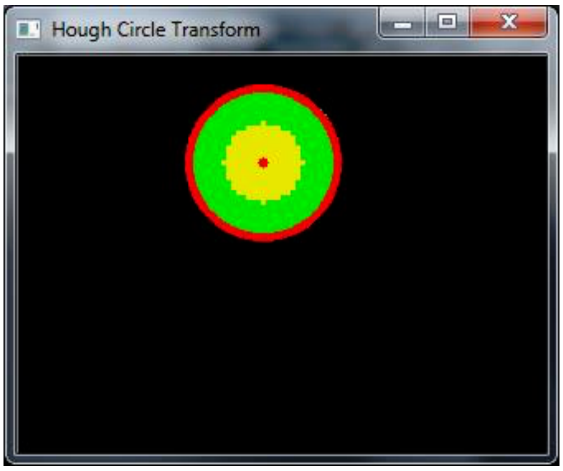


Figure 8: Hough Circle Transform detection

Controlling the movement of the icub head

The iCub's head needed to be controlled with the program in order to follow the circle that was visible on the screen. In order to control the iCub's head, the ikinGaze controller was used. Rescaling was needed to go from windows view which is 240x320 to ikinGaze view.

ikinGaze takes three parameters:

1. The x axis: looking into the distance (negative number). I.e. -3.0 is looking 3 m in the distance.

2. The y axis: looking left and right (chose values from -0.25 to 0.25)
3. The z axis: looking up and down (chose values from 0.2 to 0.6)

Problems Overcome

There were several issues the were addressed during this project.

- **OpenCV with Yarp:** There were issues setting up the cmake file correctly. This was needed in order to openCV and yarp working simultaneously. The solution involved adding the include and link directories for both openCV and Yarp.

- **Writing to Yarpview:** When trying to send the image back to yarpview it was a struggle know how to write an image back to the port. In the end the output port was prepared with the yarp image which could then be written to the port.

- **Colour:** Some times the image wanted RGB and other times it wanted BGR. When reading the image from the camera it was read in as Rgb. However, when writing the image back to the yarpview it was written as a Bgr. This is due to colour conversion when changing between Yarp and openCV.

- **Circle detection:** Circle was not perfect. As a result, occassionaly it detected more than one circle resulting in two centres just off from each other. This could have been solved by playing around with the variable values more to achieve a single circle each time.

- **Detecting head movement:** The icub's head occasionally wobbled and the movements were not large obvious ones. This made it a struggle to figure out whether the robot was actually focusing on the circle or just wobbling its head. When a sleep was put in, to make sure the iCub moved its head fully, the movements became more obvious