# README

These are some notes on the code for the paper “Stereopsis without correspondence” by me, Jenny C. A. Read, in the issue of Philosophical Transactions of the Royal Society B on “New approaches to 3D”.

The code was run in Matlab 2020b, using the Deep Learning Toolbox and, er, possibly others.

## Coordinate system

We define world-centric Cartesian coordinates (*X,Y,Z*) and use vector to represent a unit vector along the X axis, etc. In this paper, the beast does not move position but remains at the origin, but to be fully general we will use vector **B** (code: Beast.HeadPosition) to represent the beast’s position in the world-centric coordinate system. The beast’s position is defined to be the location of the mid-point between its two eyes, which are a distance *I* apart. In this paper, ***B****=0* and *I*=1cm.

The rotation matrix MW specifies the pose of the beast’s head relative to the world-centric axes. We define this using azimuth-longitude αW and elevation-latitude κW (Fick coordinates):

Equation

The nodal points of the left, right eyes are at

where the + holds for the left eye and – for the right, and *I* is the interocular distance.

The gaze vector **g** (code: Beast.GazeVector) defines where the beast is looking:

If αW = κW = 0 , MW becomes the identity matrix. Then, the beast’s interocular axis coincides with the world-centric *X* axis and the gaze vector with the *Z* axis.

## Files

See also section Pipeline below.

### SimpleBeast.m

Defines properties and methods for the simple beast. The beast has a head whose location in space is defined by the 3D vector. HeadPosition, corresponding to **B** in the paper. In this paper, the head remains at the origin; it does not translate.

The head can rotate, though, and its current posture with respect to the world-centred axes is specified by .HeadAzimuth and .HeadElevation.

.GazeVector is a unit vector defining “straight ahead” for the beast. When .HeadAzimuth and .HeadElevation are 0, .GazeVector looks along the Z axis.

The eyes are not converged at all, so the optic axes of both eyes are always parallel to .GazeVector.

Beast.GetAzElDist(Position): Position is a 3xN array specifying the (X,Y,Z) coordinates of N points. GetAzElDist returns the azimuth-longitude, elevation-latitude and distance of each point relative to the origin, after rotating them so that the z-axis lies along the gaze vector

[RetinalImages,newAz,newEl,jnew] =Beast. ProjectScene(Objects) :

Note that this uses functions from Matlab’s Deep Learning Toolbox, e.g. fullyConnectedLayer.

Also notice that when you make a Beast, it comes with a figure Beast.RetinaPlot. Whenever I project a scene onto retinal images, I render it onto this figure. It’s not sophisticated but it works – kinda!

### TrainBeast.m

File to run when you want to train a beast.

### Make3DscenesConstAngSize.m

Makes a large number of images for training. You need a “Beast” object to do this, because for each 3D scene, we are going to project this onto the beast’s retinae and tell it where it should fixate next (using Beast.ProjectScene).

### BeastDemo.m

Loads a trained beast and makes a video of it turning its head as objects move around it in an arena.

This uses functions DrawArena.m and Timestep.m

### Timestep.m

Given a bunch of objects moving around a cubic arena which is bounded at +/- ArenSize/2 in X,Y,Z. It updates each object’s position given its speed, and has objects bounce off each other or the walls if necessary.

### AssessPerformance.m

As you might guess, this was me testing out the trained Beast’s performance in various situations. I make scenes consisting of two objects at +/-15deg azimuth, one 4cm from the Beast and one at 10cm. Obviously it should always choose the closer object, at 4cm. I generate 500 noisy images, adding Gaussian noise with an SD of 0.2 to the images that are otherwise binary – 1 where there is an object and 0 where there is not. I plot histograms showing the chosen direction for 3 different situations: (1) Both objects have the same physical diameter (1.1cm), so the closer object subtends a larger angle; (2) Both objects subtend the same angle (15deg), so the closer object is smaller; (3) the closer object is physically so much smaller that it also subtends a smaller angle (10deg as compared to 15deg for the larger object).

I then go on to examine the response for vertical disparity, and to generate the plots shown in Figure 12 of the paper.

### PairExperiments.m

Again assessing performance. This generates Figure 11BC in the paper.

### Fig\_ShowWeights.m

Was used to generate Figure 10 in the paper. It begins by loading TrainedBeast.mat, so this needs to be available.

### Fig\_Projection.m

Not important – this was me checking a few things out. First, I plotted 8 objects located around the clock in front of the Beast, 5 units along the z axis, each 20 deg in diameter. I projected them onto the model retina to see how they were distorted. Then, I loaded StupidStereo3D\_51x51\_4Object10deg\_100000G.mat and drew a random scene. This is what I did in Figure 18B in the paper, although with a different example image.

# Pipeline

For the paper, I first ran Make3DscenesConstAngSize.m to generate noise-free retinal images for the 100,000 training scenes, along with what would have been the correct direction for this scene (i.e. to fixate the nearest object). This file also calls on SimpleBeast.m to define the Beast, so that we know how to make the retinal images. These retinal images were saved in StupidStereo3D\_51x51\_4Object10deg\_100000G.mat

The relevant part of the Methods is this:   
“For training, we generated 100,000 noise-free scenes like the example in Figure 18 (Matlab file Make3DscenesConstAngSize.m in the code repository). Each scene contained 4 spherical objects with azimuths and elevations generated independently from a normal distribution with mean 0o and SD 45o. Distances were generated independently from a uniform distribution over the range 1 to 11cm. Diameters were scaled so that each sphere subtended 10o at the origin. Since the model retinas extended only to ±70o eccentricity, it was possible for some spheres to be out of view. For classification, the correct output unit was defined to be that representing the headcentric direction closest to that of the nearest visible object (strictly, the nearest object whose centre was visible). If no objects were visible, the correct unit was that representing (0,0), so that the head would stay in its current position.”

I then ran Make3DscenesC.m to generate noise-free retinal images for 500 validation images. Here, there were only 2 objects and they could be anywhere from 0.1 to 20 deg in diameter. There were some other differences as well but the details aren’t important since these images didn’t affect the training .They were used only for validation so I could get a sense for whether the model was performing OK on these images which were not used in training. These images were saved in StupidStereo3D\_51x51\_2O\_500.mat .

I then ran TrainBeast.m. This loads in the training and validation images, and trains the network to try and optimise the number of correct direction choices in the training dataset.

I then ran AssessPerformance.m and PairExperiments.m for some quantitative metrics.

To make a video to get a more informal sense of how the Beast was performing, I ran BeastDemo.m