# Computer Vision on the Raspberry Pi: Part II

Last time, we saw how PiTeR the terrestrial robot can identify symbols he sees in his environment, even when the symbols are at an angle to the ideal symbol being used as comparison.

## Take me for a drive

Now we want to enhance our example to distinguish between the different symbols and act on them. PiTeR needs to turn and drive based on what he is seeing. Because the actions and control logic are are different, we’ll deal with turning and driving separately.

PiTeR balances on two wheels. How he does this is not part of this article, but you can find out more on his website. He drives forward or backward when both wheels turn in the same direction and at the same speed. When the wheels rotate at different speeds, he turns. If he is standing still and the wheels rotate in opposite directions, PiTeR turns on his own axis.

So now let’s think about what we want to get PiTeR to do. We want him to look in his visual field for a symbol of a particular colour. What we want to do next depends on how far away from the symbol he is. If he is far from the symbol, we want him to drive towards it. If he is close we want him to recognise the symbol and act on it.

You may recall from the last article that the symbols mean ‘turn left’, ‘turn right’, ‘not this way’ and ‘home’. It’s clear what we want to do for the ‘turn’ symbols, but what about ‘home’ and ‘not this way’? In the case of ‘not this way’, we want him to turn around and go back the way he came. In the case of ‘home’, he is at the end of the trail and has reached his goal, so let’s make him do a little victory dance.

So first of all, we need to decide if we are near or far from the symbol. This is easy to do, from last time, you will recall that we get a rectangle locating the symbol patch in PiTeRs visual field. All we need to do is to compute the area of the rectangle. If it is less than a certain size we will drive, if it is more, we will stand still and figure out which symbol it is.

Once we are close enough, we will use the ‘good match’ features OpenCV found for us as we discussed last time. The symbol with the greatest number of good matches will be the one we act on.

Before we go off and start work on this, there are a couple more things to think about. Once we start driving around, we can’t go off into a loop and forget about everything else. PiTeR constantly exchanges information with the person controlling him; for example, they might decide to intervene while he is doing his automated actions. If he starts to ignore his human, he’ll be branded a rogue robot!

He also needs to exchange information with his wheel control system in order to drive and equally important, to know how far he has gone and slow down when necessary.

To support these needs, PiTeR executes his long running algorithms, like symbol detection, on separate processing threads. Only when the image processing is complete do we look at the results in the main drive loop. This keeps PiTeR nice and responsive to human control.

Next, our previous script read just one example symbol and looked for a match with it. We now need to recognise four different symbols, so we’ll need to adjust the script to read all four symbols and compute the key image features for each one. As before, we’ll do this at start up time because it only needs to be done once for each reference image.

We’ll also need to adjust our matching algorithm to loop over the four sample symbols and compute the good matches for each one. So the script will look like this:

# Repeat

# Find a patch

# If the patch is too small:

# drive towards it.

# if the patch is not in the centre, turn while driving to bring it more towards the centre

# else:

# crop the image to the size of the patch (actually we’ll crop a bit bigger than this)

# loop for each symbol

# compute the good matches for the symbol

# remember the symbol with the most good matches

# if the matched symbol is a turn command

# turn left or right 90 degrees

# else if the matched symbol is ‘not this way’

# turn through 180 degrees

# else if the matched symbol is ‘home’

# do a dance and exit the outer loop

In all cases except the ‘home’ case, PiTeR will find the next patch in the trail. It will be further away, so he will drive to it and the whole outer loop will be repeated until he reaches the ‘home’ symbol. Yay, PiTeR can do cub scout trail tracking!

## More computer vision tricks

Following symbols is not the only thing PiTeR can do. OpenCV also has facilities for detecting and recognising faces. To do this we use an OpenCV CascadeClassifier. This can be programmed to look for the classic shape of a human face. This is the same method that modern cameras use to choose where to focus. If one is found in the field of view, we command PiTeRs head servos to try to centre the face in the field of view. This gives him the ability to follow you as you move around.

If more than one face is found, we choose a face at random and centre that. This gives the uncanny impression that PiTeR is looking at the different people in a crowd (and in a way, I suppose he is).

We can take this one step further and have him say hello to individual people he recognises. To do this, we train him with still photos of the people we want him to say hello to. We’ll discuss face recognition in the next article.