## Vision (Shaheen)

The objective of the Vision Subsystem is to use the camera on the terasIC D8M Digital Camera module to observe the environment and discover or identify objects that could be in the path of the Rover. These obstacles take the form of multi-coloured balls (aliens) or a striped piece of paper (a building).

After identifying these objects, this Subsystem will determine the Rover’s approximate distance away from the obstacles as well as relative orientation and communicate this with the Command and Control subsystems so that they can be safely avoided and displayed on the Rover’s web interface.

To accomplish this, the Rover must make use of an object detection algorithm, specifically designed to target the balls and the striped paper; this is accomplished in a series of steps listed below.

### 1 HSV Conversion

To make image processing more practical, the implementation of an RGB to HSV converter was necessary – colour itself, or Hue, is represented as a single parameter ranging from 0-360, and intensity and brightness are represented as a combination of Saturation and Value (ranging from 0-255 in this specific implementation to avoid floating point calculations and the performance/space trade-offs that come with it). This representation allows easy manipulation of the image as the object detection algorithm works by simply detecting if the values of the current pixel fall into a defined threshold for each of the known obstacles.

[RGB to HSV conversion | color conversion (rapidtables.com)](https://www.rapidtables.com/convert/color/rgb-to-hsv.html)

[HSV to RGB (and back) without floating point math in Python - Stack Overflow](https://stackoverflow.com/questions/24152553/hsv-to-rgb-and-back-without-floating-point-math-in-python)

Talk about specific implementation

### 2 Auto Gain Algorithm

To detect obstacles in the Rover’s environment, the parameters of the camera must be set so that it does not cause extreme bright spots in the image whilst maintaining a clear and well-lit image. The parameter to be considered is the camera’s gain, which ranges from 0-800 and directly affects the brightness of the image seen by the rover.

During the HSV conversion of the image, the running sum of the values across all pixels is maintained and communicated to the NIOSII processor at the end of the frame – as Value represents the ‘brightness’ of the pixel, a running total of the average Value was tracked, and then by the total number of pixels at the end of the frame. If the average Value is deemed too low or high, the algorithm simply adds or subtracts a small unit to the current camera Gain, which then increases or decreases the Value, eventually stabilising it - this is essentially a negative feedback controller. Using this algorithm, the object detection algorithm can be assured the current image is bright enough to identify any objects and does not require specific tuning for different brightness levels.

### 3 Colour Separation and Noise Removal

The method to capture the obstacles simply involved defining HSV thresholds for each obstacle and detecting whether the current pixel falls into the defined threshold – if it does, then it is considered to be part of that obstacle and will be accounted for during distance and relative angle calculation (the ultimate goal of this subsystem).

The values used to define the thresholds for the object detection was made through trial and error as well as the usage of our own Python script (named HSVBoundHelper.py) that easily allowed us to view the effect of different lower and upper thresholds on any given image instantaneously, rather than adjusting the values in Verilog and waiting for it to compile to see the effect:

|  |
| --- |
| A picture containing orange, fruit, close  Description automatically generatedGraphical user interface, website  Description automatically generatedAn example of the usage of the HSVBoundHelper.py script that removes pixels that does not fall within the upper and lower HSV values inputted by the user. The original image is shown on the right. |

|  |
| --- |
| Background pattern  Description automatically generatedA screenshot of a computer  Description automatically generated with medium confidence [Full Color Spectrum - ColorTools.net |](https://www.colortools.net/color_spectrum.html)  A colour spectrum image was also used to see exactly what colours the bounds defined. |

#### Technique 1

Originally, a posterization/quantise algorithm was used to reduce the colours available to the object detection algorithm in order to better ‘pick out’ the different obstacles as they would lie in their own unique quantised bands for the H, S and V values, meaning no filtering would be needed.

This was done in at the same time as the noise filtering above, the filtered pixel is then quantised/posterized through the following technique:

1. Separate H, S and V into a discrete number of steps
2. Determine the bands in between which the values for that pixel lie.
3. If the actual value of the pixel is above the midpoint of that band, set it to the upper end, else, set it to the lower end.

|  |
| --- |
| A screenshot of a video game  Description automatically generatedAn example of how noise in the image was made worse by the posterization algorithm. |

Once implemented, the image noise became much more apparent as random pixels would fit into the thresholds defined for the balls, causing the image to look even worse than before (as noisy pixels were now highlighted instead of being filtered out). This made the algorithm redundant and was an oversight in retrospect as the noise is inherently random and would not conform to the pixels neighbouring it.

The steps then taken to mitigate the image noise, which takes the form of Gaussian noise and salt and pepper noise, pre-posterisation are described as follows:

|  |
| --- |
| Graphical user interface, application  Description automatically generatedFlow Summary after Median filter implementation |

Both types of noise could be attenuated using a median filter – a 3x3 window is constructed around the current pixel, and the median pixel values of the 9 are selected for the values of that current pixel. This requires an extraordinary number of resources as 3 rows of pixel data must be kept in memory, which equates to a 2D array of 640 \* 3 containing 25 bits each (for the HSV values). Even after heavy optimisation of the algorithm used, the algorithm caused the logic cell usage of the FPGA to exceed 100% and was then discarded because other steps still needed to be implemented and it was likely not possible even with more optimisations.

An alternative approach was to use a 9-pixel median filter that was restricted to only the X axis, but as the median filter is not a linear filter, it would not produce the same results as the 3x3 window median filter approach.

This algorithm was also computationally expensive as it required several comparisons across the three different HSV parameters for the individual bands to perform the quantisation, as well as a sorting algorithm in order to determine the median value from an array of size 9 three times.

#### Technique 2

Taking inspiration from this attempt, a new algorithm was devised which by nature performs filtering, and therefore noise removal, whilst also requiring much greater tolerances in the HSV thresholds defined for each obstacle.

The revised technique is described as follows:

1. Allocate a shift register of size n for every object to be detected.
2. If the current pixel activates a pixel threshold defined for any of the obstacles, store a 1 at index 0 into the shift register for that obstacle.
3. For the current pixel, if the number of 1s in the shift register for any obstacle exceeds threshold k (this limit can be tweaked), consider the current pixel to be part of the obstacle, and consider it for distance and angle calculation (described later). Additionally, set the pixel’s colour to be that of the obstacle it is detecting – i.e. if there are at least k pixels that are detected as red in the n pixels before the current pixel, the current pixel will be outputted as red. This is the filtering step, and essentially is a mode filter.
4. Shift all shift registers by 1 bit.

The mode filter defined in step 3 is the step that cements this algorithm as being vastly superior to the previous one; as it is a mode filter, no sorting algorithm is required, and is easily scalable as only the length of the shift registers is the only parameter that would change – allowing for easy sensitivity adjustments.

Additionally, the mode filtering removes a substantial amount of noise from the image as the other n-k pixels in the window would vastly outnumber the number of noisy pixels – adjustments to the k threshold or the n window could vastly improve noise mitigation but may also introduce false positives, however as described in the next step, noisy pixels that happen to fall into the HSV bounds for the obstacles will not affect the algorithms ability to detect them, so parameters that optimise entire detection of the obstacles are preferred over ones that mitigate noise.

### 4 Bound Drawing

From the mode filtering above, the next step is to draw vertical bound lines around the obstacles to measure the distance of the obstacles from the camera. This bound drawing must only contain the obstacle, which means it must be insensitive to noise pixels that happen to fall within the colour detection ranges for the different obstacles.

This algorithm works as follows:

1. If the mode filter has determined the current pixel to be part of an obstacle, and it is the first pixel to be detected, set it as the temporary left side bound.
2. From now on, if more pixels are detected, keep count of how many pixels have been detected thus far, ie the width, and set the temporary right side bound to the most recently detected pixel.
3. If a pixel is no longer detected, assume the entire obstacle has been scanned over. If the width of this specific colour obstacle is larger than anything detected thus far, replace the true bounds with the temporary ones (this step is what neglects the noise, as noisy pixels would only be a few pixels in width and thus cannot override an obstacle if it is within range of the camera).

To further combat noise, a minimum distance between the left and right bounds is enforced at 60px.

A screenshot of a video game

Description automatically generatedThe image on the right showcases the algorithms’ ability to reject noise – the red jumper (the noise) matches the HSV bounds for the red ball, however the red ball in the image overrides the jumper as it has more red pixels in succession. It can also be seen that the red mask around the ball is offset slightly in the x direction

as mentioned previously.

#Include screenshot of balls with bounds

### 5 Distance and Angle Calculation

Originally, an analytic approach using formulae were considered to determine the distance as well as the relative angle of the obstacle, however lack of knowledge about the parameters defining these equations as well as the fact that they are not static, meant that it would be difficult to implement.

Instead, an approximation was used with exponential regression – we simply measured the actual distance away from the camera the obstacles were and the apparent size of the obstacle in pixels and plotted the data. This allowed us to find a best fit curve to use for the rover’s distance estimation.

|  |  |
| --- | --- |
| Pixel Width (px) | Distance Measured (cm) |
| 324 | 7 |
| 195 | 13 |
| 142 | 17 |
| 107 | 25 |
| 83 | 31 |
| 65 | 41 |
| 45 | 57 |
| 38 | 71 |

[Exponential Regression Calculator (omnicalculator.com)](https://www.omnicalculator.com/statistics/exponential-regression)

After using an online exponential regression calculator, the curve had a best fit equation of y=abx where a = 68.23 and b = 0.9923. This function was implemented in the ESP32 as it requires the use of exponents and floating point numbers, the Vision system only communicates the apparent obstacle width in pixels.

As the current pixel must have k out of n pixels detected before it, the bounds drawn for the obstacles will be off by at least that same threshold k in the x direction (but the width of the detected obstacle should remain the same) – while not an issue for distance calculation, introduces an error for the angle calculation, however as the angle data lacks precision, the relatively small error (compared to the 480 horizontal resolution) is insignificant and can be ignored.

Angle calculation is fairly trivial – it is defined as the midpoint of the obstacle/20 (which will map the 0-480 horizontal resolution to 0-31, which fits the 5 bit data width at the end of a packet.)

### 6 SPI Communication with ESP32

[ESP32 Pinout Reference: Which GPIO pins should you use? | Random Nerd Tutorials](https://randomnerdtutorials.com/esp32-pinout-reference-gpios/)

[SPI Core, Quartus II 7.1 Handbook, Volume 5 (intel.com)](https://www.intel.com/content/dam/support/us/en/programmable/support-resources/bulk-container/pdfs/literature/hb/qts/n2cpu-nii51011.pdf)

[UART and SPI on FPGAs - FINCH - Confluence (utat.ca)](http://spacesys.utat.ca/confluence/display/FIN/UART+and+SPI+on+FPGAs)

Diagram of packet layout

After unsuccessfully trying to use the Intel 3-Wire SPI IP Core, a 3rd party guide that uses the alternative Avalon-ST SPI Intel IP Core was followed, which implemented the necessary hardware required to communicate with the ESP32; however, the core only supported data widths of 8 bits, which meant that the expected 16-bit data packet for each obstacle had to be broken down into 2x8 bit packets.