Digital and Embedded Systems
Associate Lecturer Kieran Quirke-Brown

Solutions 9 – Image Processing and data transmission

1. Using the Robios library detect if there is a moving red object between two images. Determine if the movement occurred in the left, right or middle section of the image.

This involves a number of steps

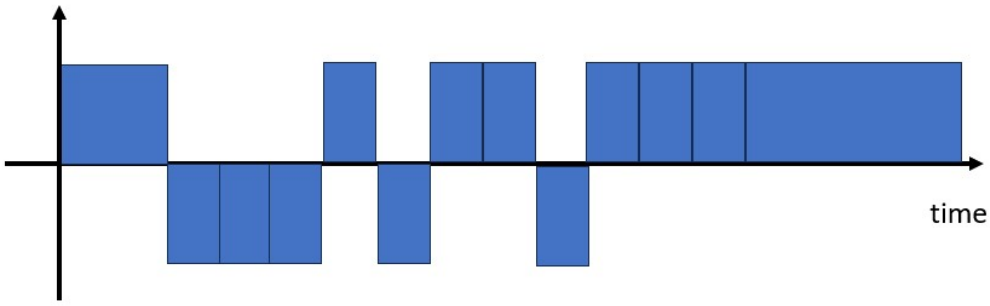
- **If not supplied, capture 2 images (need to be spaced apart in time)**
- **Find the red values in both images**
  - **Convert to HSI**
  - **Use a threshold to convert to binary**
- **take the absolute difference between the two images**
- **count the number of non zero pixels in each region**
  - **if that number is great than some threshold then there is movement**

start by writing some functions first:

```
4   void getBin(BYTE* imgIn, int* outIm) {
5       BYTE* h, s, i;
6       hsiIm = IPCol2HSI(imgIn, h, s, i);
7       for(int i=0; i < 160; i++) {
8           for(int j=0; j < 120) {
9               if(h[j*120 + i] > 42 - 5 && h[j*120 + i] < 42 + 5) {
10                  outIm[j*120 + i] = 1;
11              } else {
12                  outIm[j*120 + i] = 0;
13              }
14          }
15      }
16  }
17
18
19  void diff(int* imgIn1, int* imgIn2, int* outIm) {
20      for(int i=0; i < 160; i++) {
21          for(int j=0; j < 120) {
22              outIm[j*120 + i] = abs(outIm2[j*120 + i] - outIm1[j*120 + i]);
23          }
24      }
25  }
26
```

```
void main() {

    CAMInit(QQVGA);

    while(1) {
        BYTE* im1, im2;
        CAMGet(im1);
        OSWait(20); // wait 20ms to capture a second image
        CAMGet(im2);

        int* bin1, bin2, diff;

        getBin(im1, bin1);
        getBin(im2, bin2);

        diff(bin1, bin2, diff);

        int countL=0;
        for(int i=0; i < 53; i++) {
            for(int j=0; j < 120) {
                if(diff[j*120 + i]) {
                    countL++;
                }
            }
        }

        int countM=0;
        for(int i=53; i < 105; i++) {
            for(int j=0; j < 120) {
                if(diff[j*120 + i]) {
                    countM++;
                }
            }
        }

        int countR=0;
        for(int i=105; i < 160; i++) {
            for(int j=0; j < 120) {
                if(diff[j*120 + i]) {
                    countR++;
                }
            }
        }

        if (countL > 30) {
            LCDPrintf("movement on the left\n");
        }
        if (countM > 30) {
            LCDPrintf("movement in the middle\n");
        }
        if (countR > 30) {
            LCDPrintf("movement on the right\n");
        }
    }

}
```

2. For the given image determine the hex value sent. Look up what ASCII value has been sent. The message sent has 1 start bit, 1 odd parity bit and a stop bit. Determine if the parity bit is correct. If the bits are to be sent at a baud rate of 115200 determine how often a bit need to be sent and write some c code to send this.



Values are sent LSB first so the sent value is 10110100 which is 0xB4 using the ASCII table we see that hex B4 is not used (this could mean that it is a different encoding or that the transmitted value was incorrect). We note that the parity bit is 1 and we are using odd parity. We sum up the number of 1's sent in the data section which adds to 4, this is an even number so according to this the parity bit is correct. However imagine now that what the sender actually sent was 01110100 and the two most significant bits are the opposites (this could occur due to noise or mismatched timing between devices). The hex value for this is 0x74 which represents the ASCII value of 't'. To determine how often the bits should be sent we divide 1 second by the baud rate giving us ~8.7us. Let us now write some code to send this message. We will use GPIO1 as the sending bit line.

```
void sendT() {

    digitalWrite(1, LOW); //start bit
    usleep(9);
    /*usleep takes an integer value so we can sleep for the time
    we want, there are two ways we can deal with this. First is
    to use nanosleep function but this has extra setup. Otherwise
    we can use usleep, round it to the closest value and hope
    that the delay won't cause too much issue. For this example
    we use the simpler approach but note that long strings
    will break this implementation.*/

    //sending data
    digitalWrite(1, LOW); //not necessary but helps keep track
    usleep(9);
    digitalWrite(1, LOW);
    usleep(9);
    digitalWrite(1, HIGH);
    usleep(9);
    digitalWrite(1, LOW);
    usleep(9);
    digitalWrite(1, HIGH);
    usleep(9);
    digitalWrite(1, HIGH);
    usleep(9);
    digitalWrite(1, HIGH);
    usleep(9);
    digitalWrite(1, LOW);
    usleep(9);

    digitalWrite(1, HIGH); //parity bit
    usleep(9);
    digitalWrite(1, HIGH); //stop bit to ensure we return to idle
    usleep(9);
}
```