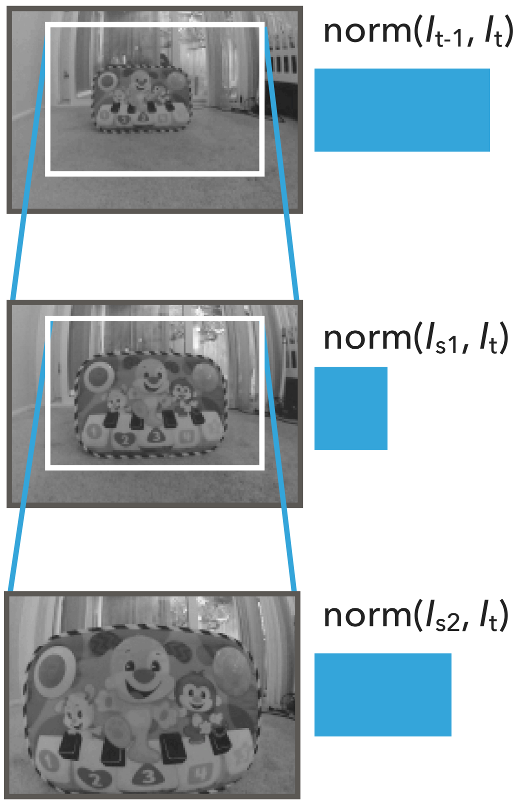
Encroachment Detection with Monocular Vision for Small, Low-cost, Compute-constrained Platforms

Visualization:

$I_{t\text{-}1}$
Previous Frame

I_{s1}
Scaled Frame 1

I_{s2}
Scaled Frame 2



norm($I_{t\text{-}1}$, I_t)

norm(I_{s1}, I_t)

norm(I_{s2}, I_t)

I_t
Current Frame

Encroachment is detected by examining a low resolution (160x120) fisheye video sequence for evidence of scene expansion. To the left, the smaller norm at I_{s1} indicates expansion.

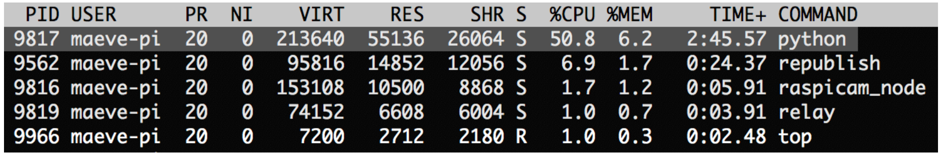
Algorithm & Complexity:

Algorithm 1 For previous and current images I_{t-1} and I_t, and scale set S, compute a scale pyramid from I_{t-1} according to S and match I_t to it using an L_1 matrix norm $\mu(\cdot)$. Let ε be a noise threshold.

```
 1: procedure EncroachmentDetection($I_{t-1}, I_t, S, \varepsilon$)
 2:     Let $\Delta_{bg} \leftarrow \mu(I_{t-1}, I_t)$ be baseline image change
 3:     if $\Delta_{bg} > \varepsilon$ then
 4:         Image change too great detect reliably
 5:         return False
 6:     end if
 7:     for $s \in S$ do
 8:         Let $I_s$ be $I_{t-1}$ expanded about its center by $s$
 9:         Crop $I_s$ to the dimensions of $I_{t-1}$
10:         $\Delta_I \leftarrow \mu(I_s, I_t)$
11:         if $\Delta_I < \Delta_{bg}$ then
12:             $I_s$ is "closer" to $I_t$ than $I_{t-1}$, this indicates
13:             that the scene is undergoing expansion
14:             return True
15:         end if
16:     end for
17:     No expansion detected
18:     return False
19: end procedure
```

Line 2 adds an $O(|I|)$ term, while the for loop at Line 7 adds $O(2|S||I|)$ due to Lines 8 & 10. Together the total complexity is $O(2|S||I|+|I|)$. A nice property of this approach is that the complexity is only sensitive to the number of scale factors and the size of the images. Therefore, in uses where both of these things are fixed, the complexity is effectively constant.

In practice, this algorithm can be implemented efficiently. An unoptimized version written in python and OpenCV with $|S| = 2$ and $|I| = 160$x120 runs comfortably on a Raspberry Pi 3, as seen below.



```
 PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
9817 maeve-pi  20   0  213640  55136  26064 S  50.8  6.2   2:45.57 python
9562 maeve-pi  20   0   95816  14852  12056 S   6.9  1.7   0:24.37 republish
9816 maeve-pi  20   0  153108  10500   8868 S   1.7  1.2   0:05.91 raspicam_node
9819 maeve-pi  20   0   74152   6608   6004 S   1.0  0.7   0:03.91 relay
9966 maeve-pi  20   0    7200   2712   2180 R   1.0  0.3   0:02.48 top
```

CPU time on a Raspberry Pi 3

Demo:

https://youtu.be/QDZJRk6OJZQ

A demonstration of the technique can be seen online by scanning the QR code or visiting the link below the QR code.

Jeffrey Kane Johnson

maeve AUTOMATION