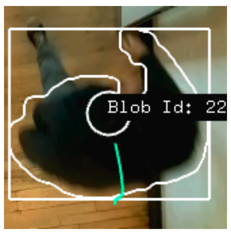
Footfall: A Camera Based People Counting System for under £60



***26/02/2017**: Footfall has recently been updated. This blog has been updated to demonstrate some of the new changes. For more updates checkout the project's GitHub Repo.*

In Watershed, we have numerous systems that tell us statistics such as how many cinema seats have been booked, how much food is left in stock and so on. But, we have no real method of capturing one of the most important metric for an arts organisation, how many people visit or are currently in the building.

This post shows how we developed a lightweight, un-intrusive tracking system with an accuracy of **90-95%** costing under **£60**.

Important!
This application does not have the ability to save live images. The images used in this post were for debugging purposes only. We are not interesting in the who, we are only interested in the how many!

Existing Systems

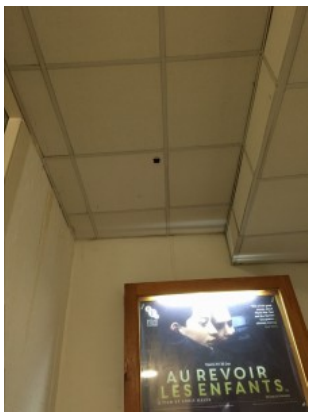
Now, we've all seen these devices sulking about in the entrances to buildings. The trusty break beam. They are able to record how many people have entered an area at any given time. These systems operate on a very simple logic: a laser is fired across a span at a retroreflective marker which reflects the beam to a reader. If the beam is broken at any point, a counter is increased.

In fairness this is a reliable method of capturing data. However, it has its problems. For instance, it is unable to distinguish how many people go through the beam at once, if two people walk side by side then they are counted as one event. More importantly they are unable to distinguish whether people are going in or out of an area (what tends to happen is a final count is taken and is divided by 2). The resulting data would then be inputted into a spreadsheet manually.

What we did?

We wanted the ability to accurately plot whether people were coming into or going out of Watershed, as well as the time they entered or left. We also wanted to see the counter statistics in real time. However, the emphasis for this project was to ensure the unit had as small a footprint as possible so as not to be physically imposing while being cost effective. Our hardware setup was a Raspberry Pi (£23), a Standard PiCamera (£20) and a PoE Unit (£10) (Power over Ethernet)*.

Fully Constructed Footfall



Footfall Mounted (yep, it's the little black box in the celling).

In terms of software there were two main programs.

The first was a openFrameworks C++ application which ran on the Raspberry Pi handling the camera input, image abstraction, tracking and people counting. In basic terms the application did the following processes.

1. Capture images from the PiCamera
2. Pass them through a series of openCV Algorithms
3. Compile blobs and track their movements
4. If blob adheres to specific conditions
5. Fire event to footfall server

The second program sat on a server and was used to contain and visualise the data from the Raspberry Pi. In this we used a combination of PHP, MySQL and Javascript.
To display the result we used the fantastic ChartJs library, which uses HTML5 <canvas> to generate graphs,charts and diagrams.

The main processes in this instance were as follows.

1. Get data from the database (MySQL, PHP and Javascript)
2. Generate Charts (Javascript)
3. Display the Charts (Javascript)
4. Update the Charts (PHP and Javascript)

In the next couple of sections we will explain some of the processes we implemented, specifically the Computer Vision algorithms and how they helped us extract people from the camera's view. We will also document how we displayed the counter data in real time using ChartJS.

OpenCV

In an ideal scenario we would have used a HoG and SVN detector to compute the tracking. For those unfamiliar with HoG (History of Orientated Gaussians) or SVM (Support Vector Machines) detectors.

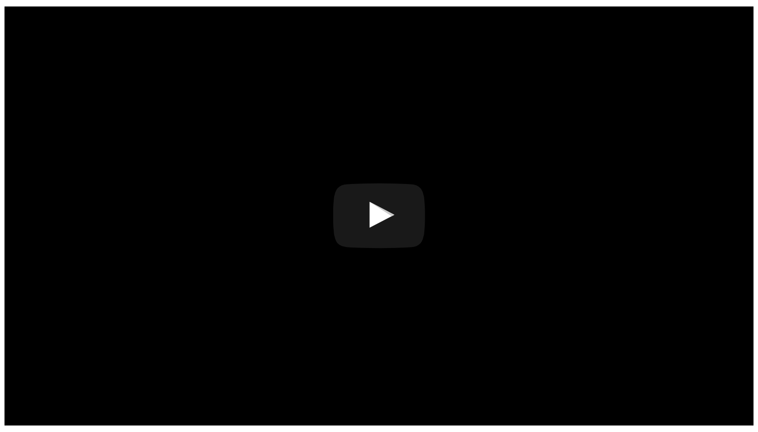
HoG is a method of object detection that counts the occurrences of gradient orientation in localised portions of an image. So HoG segments a source image into smaller blocks, calculates a histogram of each block, then calculates the orientation of each block. This source image is checked against a positively trained descriptor image. Below we see the results of applying a HoG descriptor.

These images are taken from the fantastic resource provided by Carl Vondrick and his contributors. Here is their paper Visualizing Object Detection Features

The far right image is the HoG image, this is the positive image.
If we trained the positive image to be that of a person, the HoG Descriptor might look something like this.

Trained HoG Descriptor

And if you implement the HoG to find people in a standard setting and overlay the tracker. This would be the result.



HoG Detector

However, we do not have any suitable areas in Watershed to successfully implement this system.

Instead, our approach was to use a form of blob tracking whereby the IDs of the blobs are tracked until they reach condition then they are erased and ignored in future processes.
But before any tracking could occur we needed to clean our camera image.

Preparing the Camera Image

Mask

We decided to place the counter above the main stairwell, enabling us to capture nearly all customers that entered the building (2). However, it did mean that our source image was too wide. To cut down on image processing inefficiencies, we generated a Mask Matrix that allowed us to specify the relevant sections of the image:

```
// Check whether the mask should be generated
if (_useMask)
{
        // Get the Mask Coordinates
        for (int i = 0; i < _cameraConfig.maskCoord.size(); i++)
        {
                cv::Point p = cv::Point(_cameraConfig.maskCoord[i].x,_cameraConfig
                _maskPts.push_back(p);
        }

        mask = cvCreateMat(_cameraConfig.cameraheight, _cameraConfig.camerawidth, (
        combinedMask = cvCreateMat(_cameraConfig.cameraheight, _cameraConfig.camera

        // Fill the Mat with black
        for(int i=0; i<mask.cols; i++)
                for(int j=0; j<mask.rows; j++)
                        mask.at<uchar>(cv::Point(i,j)) = 0;

        vector<cv::Point> polyright;
        approxPolyDP(_maskPts, polyright, 1.0, true);
        fillConvexPoly(mask,&polyright[0],polyright.size(),255,8,0);
}
```

After connecting to the PiCamera, images are piped into the application, these images are transferred/copied into another Matrix along with the mask Matrix; which combines both images.

```
videoMatrix.copyTo(combinedMask, mask);
```

Background

As we were using blob tracking/contour finding we needed to provide a background image to compare the live image against. This would highlight any changes to live view. But, our stairwell comes out on to an area flooded with daylight, which varies enormously. So standard background subtraction was out of the question. We therefore used a variant of the background subtraction known as MOG or Mixture of Gaussians, which essentially is like a running total of background images:

```
// Setup the Background MOG2
pMOG2 = new BackgroundSubtractorMOG2(_cameraConfig.history,
                                     _cameraConfig.mogThreshold,
                                     _cameraConfig.bTrackShadows
```

```
                                    );

// This is the ratio of the shadow detection. I.e how many times lighter the shadov
pMOG2->setDouble("fTau", _cameraConfig.shadowPixelRatio);
```

In the setup procedure you define the history length or the number of backgrounds to average. Then the threshold of the background and whether or not to find shadows.

The matrix is passed through the MOG2, it is then thresholded to remove the tracked shadows, blurred slightly then dilated and erode twice. Finally, we GaussianBlur the resulted image thresholding a final time and eroding the thresholded image which helps identify individual blobs.

```
pMOG2->operator()(combinedMask,processedMog);
// Get the background image
pMOG2->getBackgroundImage(background);
// Debug
// Copy the original MOG to the unprocessed Mat before processing
if(_showShadows) copy(processedMog,unprocessed_MOG);
// Image processing
threshold(processedMog,_threshold);
blur(processedMog,_blur);
dilate(processedMog,_dilateAmount);
erode(processedMog,_erodeAmount);
dilate(processedMog,_dilateAmount);
GaussianBlur(processedMog, processedMog,_blur*2);

// Leave these two
threshold(processedMog,50);
erode(processedMog,2);
```

Once the image has been cleaned it is passed along to the TrackingManager.

Tracking

We placed a sanity check on the contourFinder so that it only became enabled once the application had elapsed the number of history frames the MOG2 required, this stopped any false tracking data being generated on the startup of the application.

```
if(!processedMat.empty())
{
        if (ofGetFrameNum() > _historyLength)
        {
                contourFinder.findContours(processedMat);
                tracker.track(contourFinder.getBoundingRects());
        }
}
```

Counting

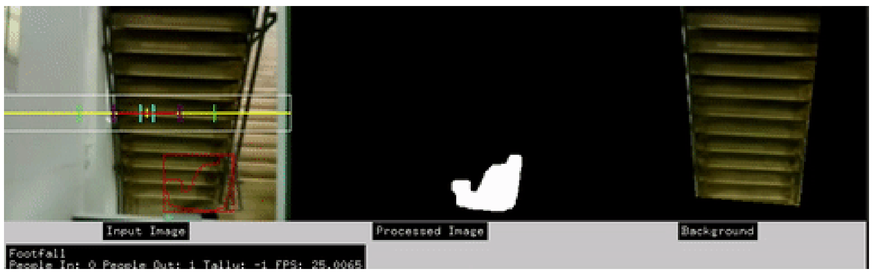
The contour finder passes the bounding boxes of the contours to the Tracker object which converts them into blobs. The next task is to count them and not just counting the number of blobs that appear, we needed more details about the blobs and their transition through our camera scene.
When a new blob is detected, our system logs it's initial position and continues to track it's current positions. Once it reaches a customisable evaluation box, the system then evaluates which direction the blob has been travelling in and final width of the blob as it enters the box.

```
vector<Blob> &blobs = tracker.getFollowers();
for(int i = 0; i < blobs.size(); i++)
{
        if (centerRect.inside(blobs[i].getCurrentPosition().x, blobs[i].getCurrentI
        {
                if (blobs[i].getCurrentPosition().y > blobs[i].getOriginPosition()
                {
                        int noOfBlobs = 0;
                        int blobWidth = blobs[i].getWidth();
                        if (blobWidth > _threeBlob)
                        {
                                noOfBlobs = 3;
                        }

                        if ((blobWidth > _twoBlob) && (blobWidth < _threeBlob))
                        {
                                noOfBlobs = 2;
                        }

                        if ((blobWidth > _oneBlob) && (blobWidth < _twoBlob))
                        {
                                noOfBlobs = 1;
                        }

                        trackingHistory.addNewData(blobs[i].getWidth(), true);
                        ofNotifyEvent(blobIn, noOfBlobs, this);
                        blobs[i].kill();
                }
                else if (blobs[i].getCurrentPosition().y < blobs[i].getOriginPosit
                {
                        int noOfBlobs = 0;
                        int blobWidth = blobs[i].getWidth();
                        if (blobWidth > _threeBlob)
                        {
                                noOfBlobs = -3;
                        }

                        if ((blobWidth > _twoBlob) && (blobWidth < _threeBlob))
                        {
                                noOfBlobs = -2;
                        }

                        if ((blobWidth > _oneBlob) && (blobWidth < _twoBlob))
                        {
                                noOfBlobs = -1;
                        }

                        trackingHistory.addNewData(blobs[i].getWidth(), false);
                        ofNotifyEvent(blobOut, noOfBlobs, this);
                        blobs[i].kill();
                }
        }
}
```

Every time a blob is evaluated an event is fired with the number of people that have just entered the building this also stops the blob from being tracked in future frames.

Depending on the settings in the configuration file, a simple HTTP Post event is created or a csv record of the event is logged. These events are logged with a the timestamp from the Raspberry Pi, where the unit is located and the number of people who have entered or left.
For the purpose of this post we'll assume that we are using the HTTP system.

The image below shows the combined systems working.



```
Footfall
People In: 0 People Out: 1 Tally: -1 FPS: 25.0065
```

- Left: Live Image with Tracking Data.
- Middle: Processed MOG Image (thresholded,blurred and dilated).
- Right: Background Image (the history from the MOG2).

ChartJS

ChartJS is a lightweight javascript library that uses HTML5 and Canvas to generate and render charts. Users can specify a number of different chart types such as pie, bar, line and polar. This section demonstrates how we used ChartJS to visualise our footfall data

Generating the Charts

We created two charts, one for displaying the total number of people in Watershed with information about the screenings and events occurring in Watershed for that day and another chart displaying the flow of traffic.

To generate the charts we first define some global variables:

```
var trafficData;
var totalsData;
var trafficChart;
var totalChart;
var interval = 300;
```

Then in a new function we create the Totals Chart. First create a variable called canvas and give it the context of the html element with an id of 'totalChart'. This looks for a space in our webpage to put the chart. Next we create a datasets object, this is the line element of our chart where we define colour, labels and data points. For our system we pre populated the data with 2 zero values and 2 timestamps 03:00 and 04:00 so that both axis have some initial data. Next we configured the chart options, by default ChartJs sets a global chart configuration, to override this simply pass your specific options into the chart object:

```
function createTotalChart()
{
        var canvas = document.getElementById('totalChart').getContext('2d');
        var datasets = [{
                label: "People In Watershed",
                fillColor: "rgba(0,77,255,.01)",
                strokeColor: "rgba(0,77,255,1)",
                pointColor: "rgba(0,77,255,1)",
                pointStrokeColor: "#fff",
                data: [0,0]
        }];

        totalsData = {
                labels: ['3:00','04:00'],
                datasets: datasets
        };

        // Chart Options
        var options = {
                // Do not animate
                animation: false,
                responsive:true,
                scaleShowLabels: 20,
                scaleBeginAtZero : false,
                skipXLabels:true,
                bezierCurveTension : 0.3,
                pointHitDetectionRadius : 10,
                scaleGridLineColor : "rgba(0,0,0,.1)",
                scaleGridLineWidth : 2,
                bezierCurve : true
        };

        // Make Chart, Pass options
        totalChart = new Chart(canvas).Line(totalsData, options);
}
```

Again for the traffic chart we create a function. Then find the canvas element with the id of ‘trafficChart’. Like the totals chart we generate the dataset, but give it two data points with different colours and labels. More importantly, when passing options to the chart we define that the bars must begin at origin which is zero and that the scale does not begin at zero. This allows us to plot positive and negative bars side by side in the same chart.

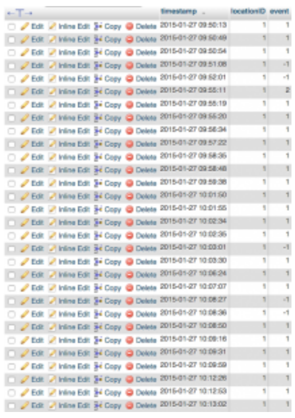
```
function createTrafficChart()
{
        var canvas = document.getElementById('trafficChart').getContext('2d');

        trafficData = {
                labels: ['03:00','4:00'],
                datasets:
        [
                {
                        label: "Traffic In",
                        fillColor: "rgba(65,190,242,0.5)",
                        strokeColor: "rgba(65,190,242,0.5)",
                        pointColor: "rgba(65,190,242,0.5)",
                        pointStrokeColor: "#fff",
                        data: [0,0],
                        chartType: 'Bar'
                },
                {
                        label: "Traffic Out",
                        fillColor: "rgba(242,65,65,0.5)",
                        strokeColor: "rgba(242,65,65,0.5)",
                        pointColor: "rgba(242,65,65,0.5)",
                        pointStrokeColor: "#fff",
                        data: [0,0],
                        chartType: 'Bar'
                }
        ]
        };

        // Chart Options
        var options = {
                animation: false,
                responsive:true,
                scaleGridLineColor : "rgba(0,0,0,.1)",
                // Important: allows the chart to draw both scales + / -
                barBeginAtOrigin: true,
                scaleBeginAtZero : false
        };
        // Make Chart
        trafficChart = new Chart(canvas).Bar(trafficData,options);
}
```

All that needs to be done is assign the correct element ids to the appropriate HTML element.

```
<canvas style='margin-left: 5px;' id="totalChart" width='900' height='300'>
<canvas style='margin-left: 5px;' id="trafficChart" width='900' height='300'>
```

Populating the Charts

When our tracker detects a person it fires an event value with a timestamp from the RPi into our database, when we select some data from the database we return the following.

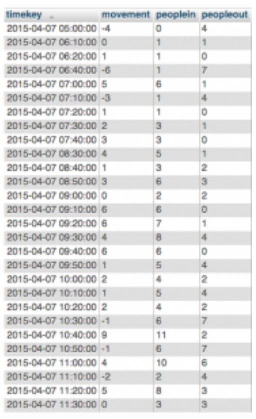


←T→	timestamp	locationID	event
☐ Edit Inline Edit Copy Delete	2015-01-27 09:50:13	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 09:50:49	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 09:50:54	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 09:51:08	1	-1
☐ Edit Inline Edit Copy Delete	2015-01-27 09:52:01	1	-1
☐ Edit Inline Edit Copy Delete	2015-01-27 09:55:11	1	2
☐ Edit Inline Edit Copy Delete	2015-01-27 09:55:19	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 09:55:20	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 09:56:34	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 09:57:22	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 09:58:35	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 09:58:48	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 09:59:38	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:01:50	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:01:55	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:02:34	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:02:36	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:03:01	1	-1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:03:30	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:06:24	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:07:07	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:08:27	1	-1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:08:36	1	-1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:08:50	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:09:16	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:09:31	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:09:59	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:12:26	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:12:53	1	1
☐ Edit Inline Edit Copy Delete	2015-01-27 10:13:02	1	1

On its own this data is fairly meaningless. Therefore, we process these events into something more tangible. We perform the following MySQL query which aggregates events into three new fields.

```
SELECT FROM_UNIXTIME( FLOOR( UNIX_TIMESTAMP( TIMESTAMP ) / 600 ) *600 ) AS `timekey
SUM( event ) AS `movement`,
SUM( IF( event > 0, event, 0 ) ) AS `peoplein`,
SUM( IF( event < 0, ABS( event ) , 0 ) ) AS `peopleout`
FROM `data`
WHERE DATE( TIMESTAMP ) = DATE( NOW( ) )
GROUP BY `timekey`
ORDER BY `timekey` ASC
```

The first field is ***movement*** this is the running total ***event*** of people in the building, so positive numbers will increment and negative number will decrease the overall count. The other two fields ***peoplein*** and ***peopleout*** evaluate whether ***event*** is positive or negative, then aggregates the result into the the corresponding field (Calculating the absolute value for the negative field). The data is then split into chunks or ranges, this can be put into a variable and changed on the fly but, for this example we have used an interval value of 600 seconds (10 minutes).



timekey	movement	peoplein	peopleout
2015-04-07 06:00:00	-4	0	4
2015-04-07 06:10:00	0	1	1
2015-04-07 06:20:00	1	1	0
2015-04-07 06:40:00	-6	1	7
2015-04-07 07:00:00	5	6	1
2015-04-07 07:10:00	-3	1	4
2015-04-07 07:20:00	1	1	0
2015-04-07 07:30:00	2	3	1
2015-04-07 07:40:00	3	3	0
2015-04-07 08:30:00	4	5	1
2015-04-07 08:40:00	1	3	2
2015-04-07 08:50:00	3	6	3
2015-04-07 09:00:00	0	2	2
2015-04-07 09:10:00	6	6	0
2015-04-07 09:20:00	6	7	1
2015-04-07 09:30:00	4	8	4
2015-04-07 09:40:00	6	6	0
2015-04-07 09:50:00	1	5	4
2015-04-07 10:00:00	2	4	2
2015-04-07 10:10:00	1	5	4
2015-04-07 10:20:00	2	4	2
2015-04-07 10:30:00	-1	6	7
2015-04-07 10:40:00	9	11	2
2015-04-07 10:50:00	-1	6	7
2015-04-07 11:00:00	4	10	6
2015-04-07 11:10:00	-2	2	4
2015-04-07 11:20:00	5	8	3
2015-04-07 11:30:00	0	3	3

```
$get = $DBH->prepare($query);
$get->execute();

if (!$get) {
        echo "Error: couldn't execute query. ".$get->errorCode();
        exit;
}

if ($get->rowCount() == 0) {
        echo "[]";
        exit;
}

$rows = array();
$runningtotal = 0;
$runningtotalin = 0;
$rowCount = 0;
$totalin = 0;
$starttime = strtotime("$endtimedate 8am");
$endtime = 0;
while ($row = $get->fetch(PDO::FETCH_ASSOC)) {
        $runningtotal += $row['movement'];
        $row['total'] = $runningtotal;
        $totalin += $row['peoplein'];
        $row['totalin'] = $totalin;
        $rows[$row['timekey']] = $row;
        $endtime = $row['timekey'];
        $rowCount++;
}
$endtime = strtotime($endtime);
if (!$endtimedate || $endtimedate == date("Y-m-d")) {
        $endtime = time() - (time() % $interval);
```

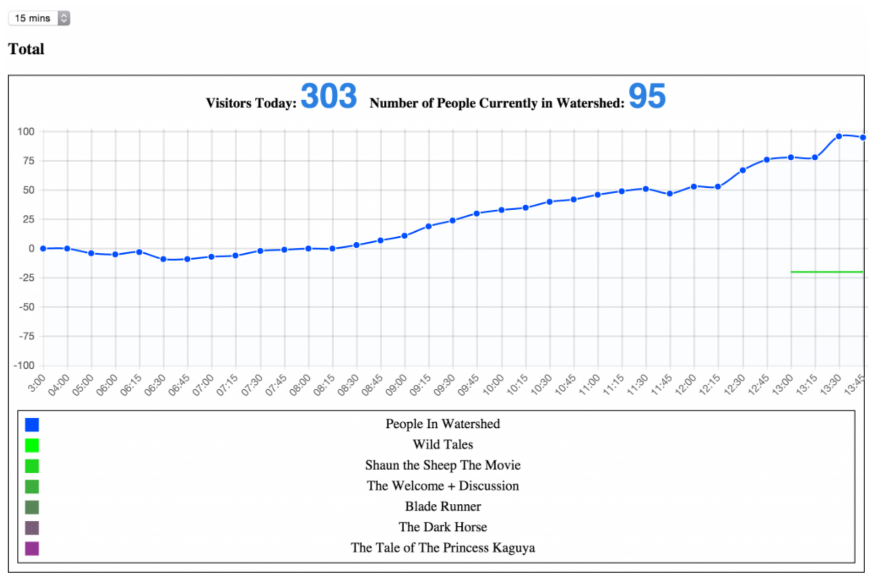
```
}
$runningtotal = 0;
$runningtotalin = 0;
for ($t = $starttime; $t <= $endtime; $t += $interval) {
        $dt = date("Y-m-d H:i:s",$t);
        if (isset($rows[$dt])) {
                $runningtotal = $rows[$dt]['total'];
                $runningtotalin = $rows[$dt]['totalin'];
        }
        else {
        $rows[$dt] = array("timekey" => $dt, "movement" => 0, "peoplein" => 0, "pe
        }
}
ksort($rows); // sort
$rows = array_values($rows); // change back into indexed

echo json_encode($rows);
```

This data is then encoded into json.

[illegible]

To input this data into the chart we make an ajax request to a PHP script on our server. The results are then parsed and assigned to the relevant chart.

```
function preLoadData()
{
        // Get the First Load of Data, this will update throughout the day
        $.ajax({ url:"http://someGetRequest.co.uk/", async: true, dataType: 'json'
                }).done(function(data){
                        // If we already have data update the labels on the web pa
                        updateLabels(data);
                        for (var i in data) {
                                labelLength++;
                                var label = data[i].timekey.substring(11,16);
                                var eventsDatasets = [];
                                eventsDatasets.push(data[i].total);
                                eventCount = 0;
                                for (var title in events) {
                                        eventAtCurrentLabel = null;
                                        var ev = events[title];
                                        for (var t in ev) {
                                                if (ev[t].start <= label && ev[t].
                                                        eventAtCurrentLabel = -10-
                                                }
                                        }

                                        eventsDatasets.push(eventAtCurrentLabel);
                                        eventCount++;
                                }
                                // Best way of getting the data into the graphs |
                                totalChart.addData(eventsDatasets,label);
                                trafficChart.addData([data[i].peoplein, -data[i].p
                        }
        });
}
```



Totals Graph Pre-populated, including film screenings that are shown as lines on the graph to help us correlate screenings with footfall

Updating the Information

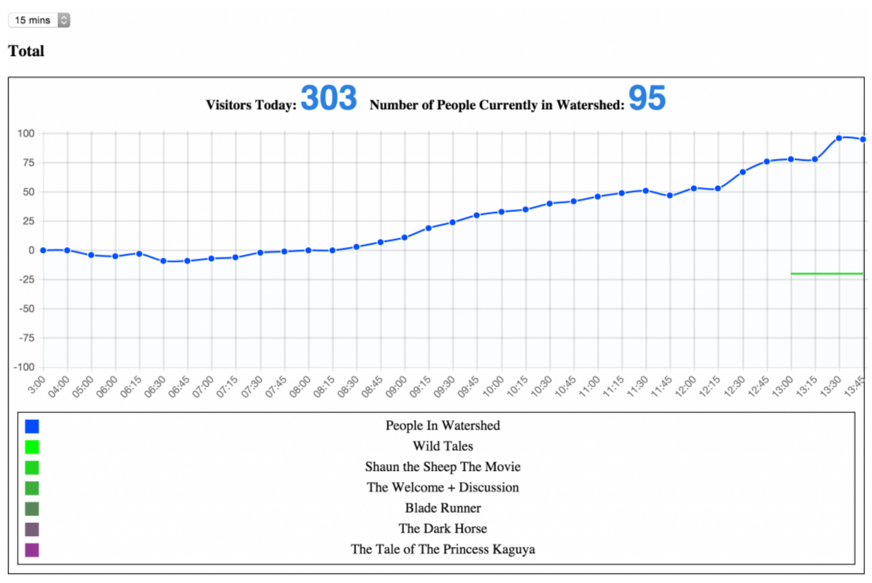
To add data to the charts after the initial load, the following function is called every 5 seconds.

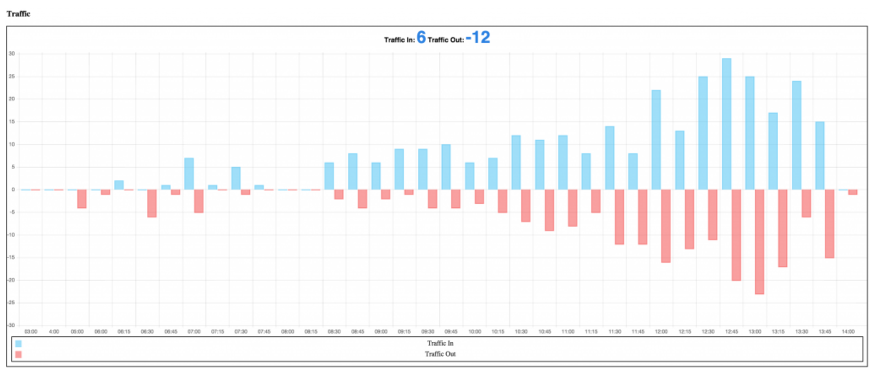
```
function updateValues()
{
        $.ajax({
                url:"http://someGetRequest.co.uk/",
                async:true,
                dataType: 'json',
                type:'get',
        }).done(function(data)
        {
                updateLabels(data);
                updateTotals(data);
                updatePeople(data);
                addDataToCharts(data);
        });
}
```

This updates the HTML and values on the page but, more importantly it adds new data to the charts. Now I stress ***new data*** because without the code below the chart would constantly add the same value until the timecode changed. So the function below evaluates the current returned timestamp against the last inputted timestamp in the charts data. If the timestamp changes then a new timestamp is added to the chart and the final value in the previous timestamp is added to that timestamp.

```
function addDataToCharts(data)
{
        if (data[data.length-1].timekey.substring(11,16) == trafficData.labels[tra
        else {
                trafficChart.addData([data[data.length-1].peoplein,-data[data.leng
                var trafficInDataLength = trafficChart.datasets[0].bars.length-2;
                var trafficOutDataLength = trafficChart.datasets[1].bars.length-2;

                trafficChart.datasets[0].bars[trafficInDataLength].value = data[da
                trafficChart.datasets[1].bars[trafficOutDataLength].value = -data[
                trafficChart.update();
        }
        if (data[data.length-1].timekey.substring(11,16) == totalsData.labels[tota
        else {
                totalChart.addData([data[data.length-1].total],data[data.length-1]
                var totalDataLength = totalChart.datasets[0].points.length-2;
                totalChart.datasets[0].points[totalDataLength].value = data[data.l
                totalChart.update();
        }
}
```

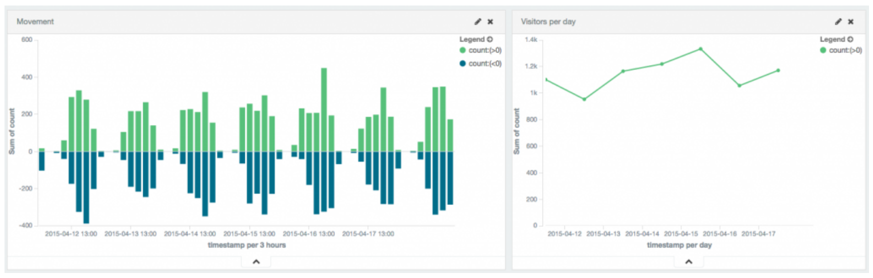
And here are the resulting charts.





Alongside putting the data into MySQL, we also made the PHP code put footfall data into our elasticsearch cluster, and experiment with graphing it in Kibana.

Although it's hard to graph the 'total people in watershed' graph, we hope it'll be a lot easier to integrate with other business intelligence metrics, such as film screenings, bar takings, conference bookings, etc.



We have made the source code from this project available on GitHub.

Disclaimer

The original software was intended for sole use within Watershed, therefore some of the source code has been altered for public use and differs slightly to our systems. For example our system generated event tags showing screening in conjunction with the total number of people in Watershed, to do this we had to pre-populate some timestamps and may cause an issue if the system is ran past a certain time.

APRIL 21, 2015 BY DAVID HAYLOCK
CHARTJS, FOOTFALL, LIVESTATS, OFXCV, OPENCV, OPENFRAMEWORKS, PEOPLE COUNTER, PICAM, RASPBERRYPI, TRAFFIC, WATERSHED

4

Search …