Do Joulies Work?

Reading an image of a LCD display with Image Classification in Mathematica

Justin Pearson

2014-10-26

Summary

Joulies are a product meant to maintain hot coffee at a comfortable temperature longer. They are sealed metal sphereoids that contain a proprietary material that melts at 140°F. The heat of fusion helps maintain the coffee's temperature. You keep them in your coffee cup and dump your coffee over them. They displace about 1.5 Tbsp per Joulie.

We took two coffee cups, one with 5 Joulies and one without, and filled them with boiling water to roughly the same level. We measured their temperatures with two digital thermometers. We took pictures of the thermometers with two different cameras: Dad's Leica handheld digital camera was set to "time lapse" and snapped a picture every minute, and my Macbook Pro took a picture about every 10 seconds.

We also had a Raspberry Pi with temperature sensors. However, water leaked into the sensors and corrupted their measurements.

Afterward, Dad manually recorded the temperature of the two thermometers from each of this 30 images.

The Macbook took 180 images. Instead of doing the obvious thing of just looking at the pictures and writing down the temperatures, I used this as an opportunity to test *Mathematica*'s ability to classify the LCD digits.

Specifically, I wanted to see if *Mathematica*'s Classify() function could identify the digits on the two LCD thermometers. I found that the classification works well for a single LCD segment (eg the one's digit) but a classifier trained on the one's digit doesn't necessarily do well on the ten's or hundred's digits, and a classifier trained for one thermometer doesn't work on another.

I haven't answered the main question -- do Joulies work -- but it was fun playing around with various *Mathematica* image-processing algorithms.

Import pictures

In[1]:=
```
Clear["Global`*"]
SetDirectory[NotebookDirectory[]]
```

Out[2]=
```
/Users/justin/Projects/Joulies/Analysis/MacbookCamera_Analysis
```

In[3]:=
```
show = Show[#, ImageSize → 200] &;
```

In[4]:=
```
filenames = (FileNames[
      "/Users/justin/Projects/Joulies/Data/MacbookCamera/MacbookCamera_Images/*.
        tiff"] // SortBy[#, FileDate[#, "Creation"] &] &);
Length@filenames
Short[filenames]
```

Out[5]=
```
180
```

Out[6]//Short=
```
{/Users/justin/Projects/Joulies/Data/MacbookCamera/MacbookCamera_Images/img
    48.tiff, <<178>> ,
  /Users/justin/Projects/Joulies/Data/MacbookCamera/MacbookCamera_Images/img
    227.tiff}
```

In[7]:=
```
timestamps = FileDate[#, "Creation"] & /@ filenames;
First@timestamps
FullForm@%
deltaT = DateDifference[timestamps[[1]], #, "Minute"] & /@ timestamps;
```

Out[8]=
```
Sun 26 Oct 2014 12:15:54 GMT−7.
```

Out[9]//FullForm=
```
DateObject[List[2014, 10, 26, 12, 15, 54.`], "Instant", "Gregorian", -7.`]
```

In[11]:=
```
{timing, rawimages} = AbsoluteTiming[Import /@ filenames];
timing
```

Out[12]=
```
6.62154
```

In[13]:=
```
show@rawimages[[33]]
```

Out[13]=

Ugh, MacBook flips it so the camera looks like a mirror. Correct:

In[14]:=
```
images = (ImageReflect[#, Left] &) @* (ImageRotate[#, 180 Degree] &) /@ rawimages;
imageDims = ImageDimensions@First@images;
```

In[16]:=
```
Manipulate[show@images[[i]],
 {{i, 33, "Frame"}, 1, Length@images, 1, Appearance → "Open"}]
```

Out[16]=

Pic description:
img 48.tiff - both reading room temp - 67deg F
img 54.tiff - begin to pour into ‘joulie’ cup
57 - begin to pour into ‘control’ cup
58 - fog on lens
227 - last pic before ‘control’ therm turns off

Get Digits of ‘Joulie’ cup’s thermometer

Trim

Crop (trim) the image to this bounding box around the Joulie therm LCD:

In[17]:=
```
Manipulate[
 trimJoulie = ImageTrim[#, {t1, t2}] &;
 {Show[images⟦i⟧,
   Epilog → {Opacity[.2], Blue, Rectangle[t1, t2]}, ImageSize → 400],
  show@trimJoulie@images⟦i⟧},
 {{i, 33, "Frame"}, 1, Length@images, 1, Appearance → "Open"},
 {{t1, {45, 410}}, Locator}, {{t2, {215, 517}}, Locator}
]
```

Out[17]=

Now we have a trimJoulie() function:

In[18]:=
```
trimJoulie@images[[100]]
```

Out[18]=

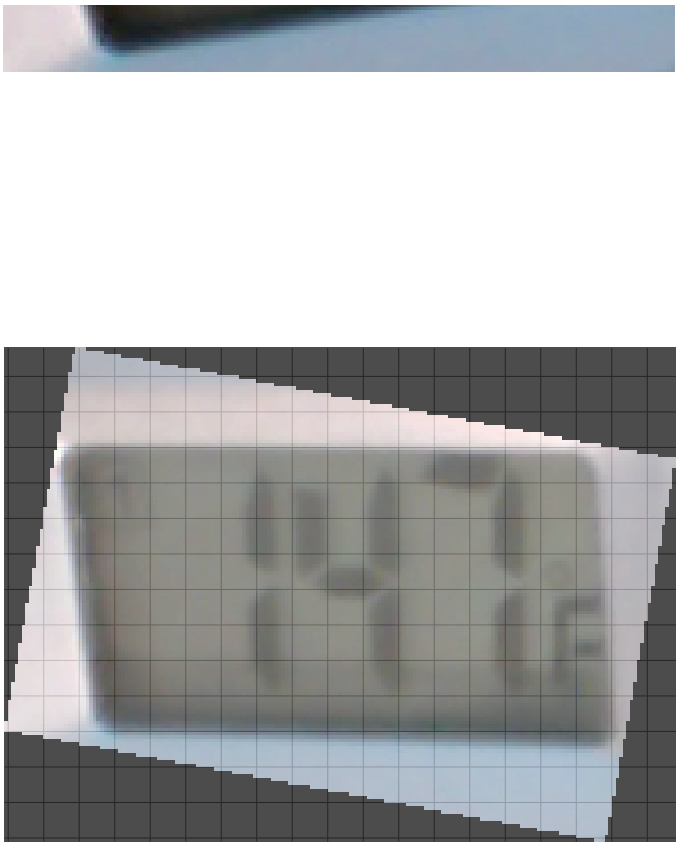
Fix Perspective

Undo the skew.

Will a simple rotation work?

In[19]:=
```
Manipulate[Module[{im, im2, dim2},
  im = trimJoulie[images[[i]]];
  im2 = ImageRotate[im, d Degree];
  dim2 = ImageDimensions@im2;
  {Show[im, ImageSize → {350, 350}],
   Show[ SetAlphaChannel[im2, .7],
    GridLines → (Range[1, #, 10] & /@ dim2), ImageSize → {350, 350}]}
 ],
 {{i, 33, "Frame"}, 1, Length@images, 1, Appearance → "Open"},
 {{d, -10.5, "Angle (°)"}, -180, 180, Appearance → "Open"}
]
```

Out[19]=



The digits seem unskewed according to the gridlines, but the edges of the screen still seem skewed.

Manual unskewing with a Perspective Transformation

In[20]:=
```
Manipulate[Module[{im, im2, dim2, err, tf, lcdCorners},
  im = trimJoulie[images[[i]]];
  im2 = ImagePerspectiveTransformation[im, {{a, b}, {c, d}}];
  dim2 = ImageDimensions@im2;
  {Show[im, ImageSize → {350, 350}],
   Show[ SetAlphaChannel[im2, .7],
    GridLines → (Range[1, #, 10] & /@ dim2), ImageSize → {350, 350}]}
 ],
 {{i, 33, "Frame"}, 1, Length@images, 1, Appearance → "Open"},
 {{a, .664}, -1, 1, Appearance → "Open"}, {{b, .152}, -1, 1, Appearance → "Open"},
 {{c, -.11}, -1, 1, Appearance → "Open"}, {{d, .68}, -1, 1, Appearance → "Open"}
]
```

Frame 33

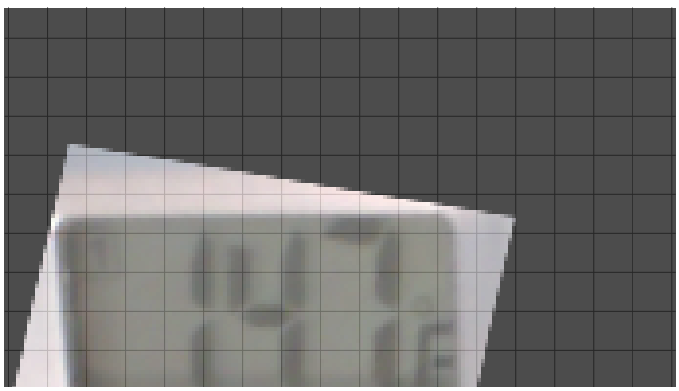
a 0.664

b 0.152

c −0.11

d 0.68

Out[20]=


,

}

Have Mathematica find a geometric transform to fix the perspective

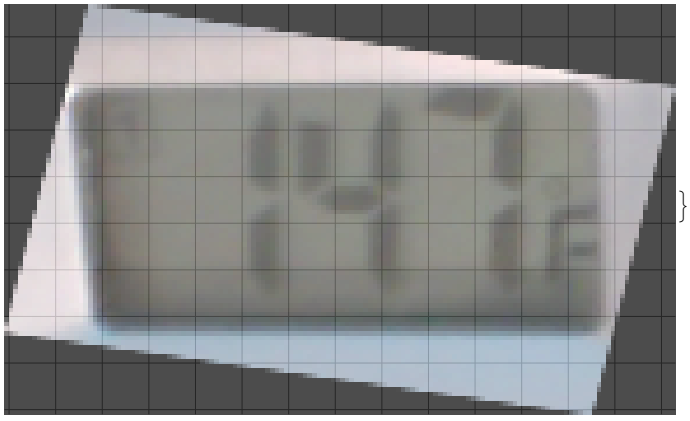
In[21]:=
```
Manipulate[Module[{im, err, tf, lcdCorners, correctCorners},
  im = trimJoulie[images[[i]]];
  lcdCorners = {c1, c2, c3, c4};
  correctCorners = {{1, 50}, {1, 1}, {100, 50}, {100, 1}};
  {err, tf} = FindGeometricTransform[
    correctCorners, lcdCorners, TransformationClass → "Perspective"];
  fixPerspectiveJoulie = ImagePerspectiveTransformation[#, tf, PlotRange → All] &;
  {Show[im, Epilog → Line[{c1, c2, c4, c3, c1}], ImageSize → {350, 350}],
   Show[ SetAlphaChannel[fixPerspectiveJoulie[im], .7],
    GridLines → (Range[1, #, 10] & /@ ImageDimensions[fixPerspectiveJoulie[im]]),
    ImageSize → {350, 350}]}
 ],
 {{i, 33, "Frame"}, 1, Length@images, 1, Appearance → "Open"},
 {{c1, {8, 79}}, Locator}, {{c2, {25, 5}}, Locator},
 {{c3, {149, 107}}, Locator}, {{c4, {169, 28}}, Locator}
]
```

Out[21]=

Frame

33

,



}

Now we have a fixPerspectiveJoulie() function:

In[22]:=
```
show@fixPerspectiveJoulie@images[[1]]
fixPerspectiveJoulie@trimJoulie@images[[1]]
```

Out[22]=

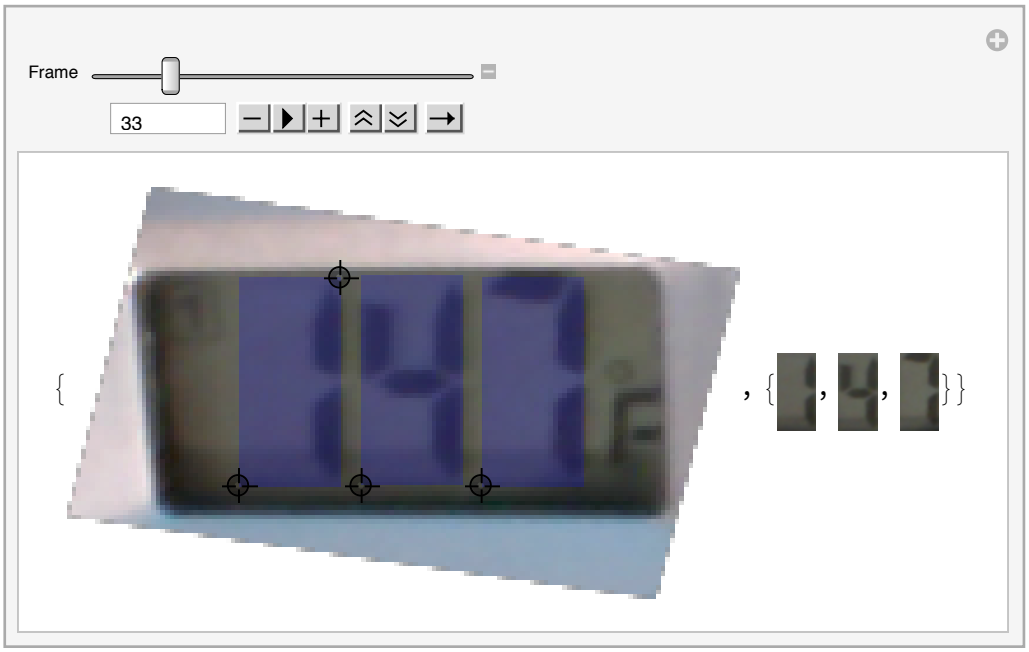
Out[23]=

Extract digits from picture

Bounding boxes around three LCD digits:

In[24]:=
```
Manipulate[Module[{im, box1, box2, box3},
  im = fixPerspectiveJoulie@trimJoulie[images[[i]]];
  box1 = {c100a, c100b};
  box2 = {c10a, c10a + (c100b - c100a)};
  box3 = {c1a, c1a + (c100b - c100a)};
  getDigitsJoulie =
   Table[ImageResize[ImageTrim[#, b], {20, 40}], {b, {box1, box2, box3}}] &;
  {Show[im, Epilog → {Opacity[.2], Blue, Rectangle @@@ {box1, box2, box3}},
    ImageSize → 350], getDigitsJoulie[im]}
 ],
 {{i, 33, "Frame"}, 1, Length@images, 1, Appearance → "Open"},
 {{c100a, {36.85`, 23.8`}}, Locator}, {{c100b, {58.5`, 68.8`}}, Locator},
 {{c10a, {62.95`, 24.2`}}, Locator}, {{c1a, {89.`, 23.8`}}, Locator}
]
```

Out[24]=



Now we have a getDigitsJoulie() function:

In[25]:=
```
getDigitsJoulie@fixPerspectiveJoulie@trimJoulie@images[[100]]
```

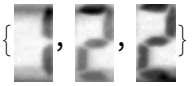
Out[25]=

Compute all digits

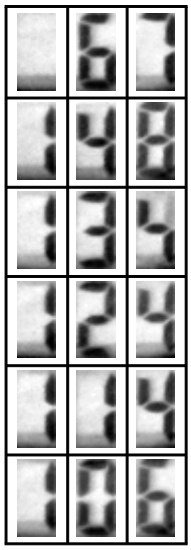
(Note: @* is 'function compose')

In[26]:=
```
digitsPixJoulie = getDigitsJoulie @* fixPerspectiveJoulie @* trimJoulie /@ images;
digitsPixJoulie =
  Map[ImageAdjust @* (ColorConvert[#, "Grayscale"] &), digitsPixJoulie, {2}];
```

In[28]:=
```
digitsPixJoulie[[100]]
```

Out[28]=


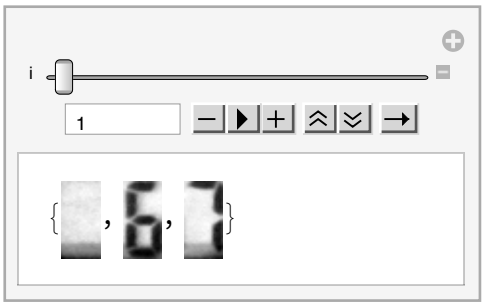
In[29]:=
```
Grid[digitsPixJoulie[[1 ;; -1 ;; 30]], Frame → All]
```

Out[29]=


Classify digits as integers

First make a training set.

In[30]:=
```
Manipulate[digitsPixJoulie[[i]],
 {i, 1, Length@digitsPixJoulie, 1, Appearance → "Open"}]
```

Out[30]=


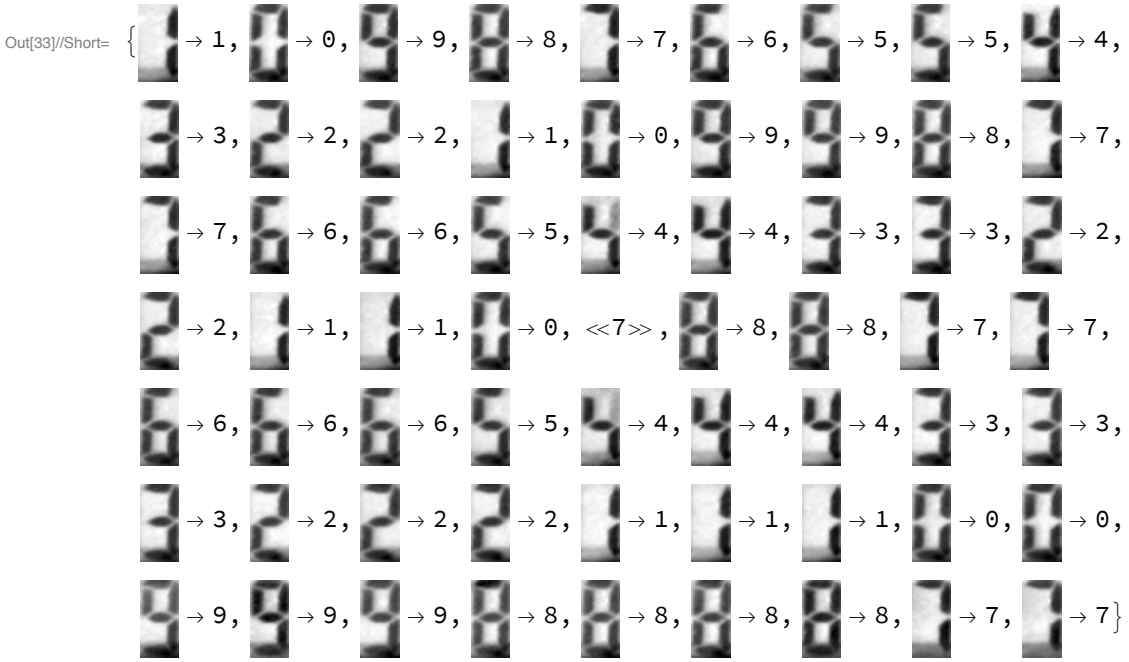
Put it on auto-play, watch it and type the numbers as they play (I put my fingers on 1,2,3,4 and 7,8,9,0). Starting at index 15, what's the least-sig digit?

In[31]:=
```
listJoulie = ToExpression /@ Characters@
    "109876554322109987766544332211000099988877666544433322211100999888877";
```

Note: Coulda just repeated this for the whole file. But if it was really long, couldn't.

In[32]:=
```
trainingDataJoulie = Table[
    Rule @@ {digitsPixJoulie[[14 + i, 3]], listJoulie[[i]]}, {i, Length@listJoulie}];
Short@trainingDataJoulie
```

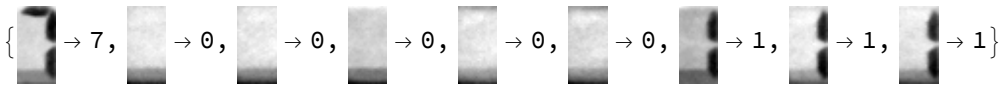
Out[33]//Short=



Manually add some mis-classified points

In[34]:=
```
extraTraining = {};
{digitsPixJoulie[[7, 2]] → 7} // AppendTo[extraTraining, #] &;
(# → 0) & /@ digitsPixJoulie[[{1, 4, 7}, 1]] // AppendTo[extraTraining, #] &;
(# → 0) & /@ digitsPixJoulie[[{-3, -1}, 1]] // AppendTo[extraTraining, #] &;
(# → 1) & /@ digitsPixJoulie[[{10, 20, 30}, 1]] // AppendTo[extraTraining, #] &;
extraTraining = Flatten[extraTraining, 2]
```

Out[39]=



In[40]:=
```
trainingDataJoulie = Join[trainingDataJoulie, extraTraining];
```

Build classifier

In[41]:=
```
{timing, c} = AbsoluteTiming[Classify[trainingDataJoulie,
    PerformanceGoal → "Quality", Method → "SupportVectorMachine"]]
```

Out[41]=

How does it work? We trained it mostly on the one's place LCD segment. Perhaps it will work on the ten's and hundred's segments too?

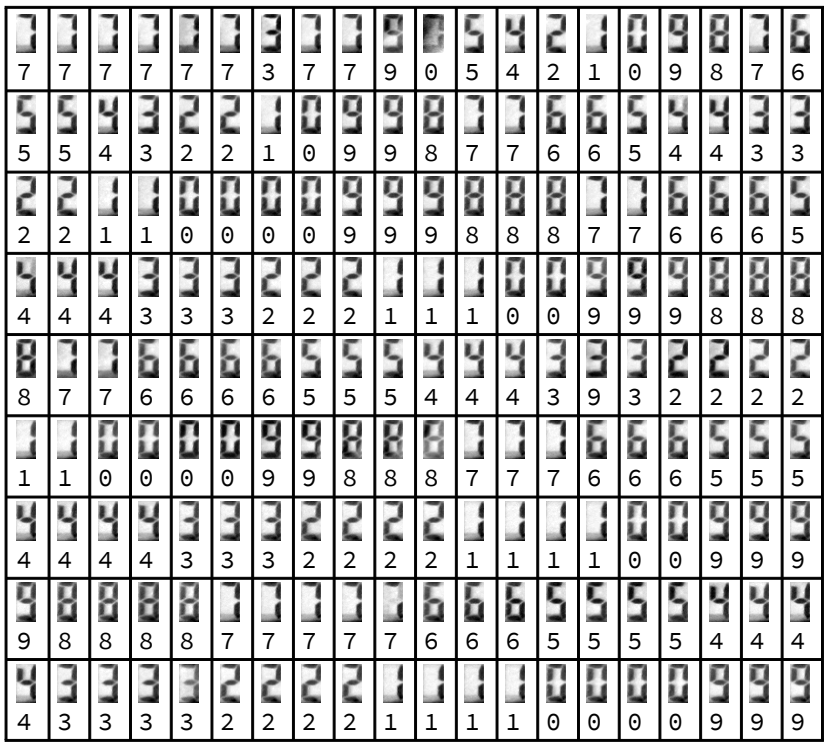
In[42]:=
```
integerListJoulie = Map[c, digitsPixJoulie, {2}];
```

In[43]:=
```
showClassifier[inlist_, outlist_] :=
 Table[Column@{Show[inlist[[i]], ImageSize → 10], outlist[[i]]}, {i, Length@inlist}] //
   Partition[#, 20] & // Grid[#, Frame → All] &
```

One's place:

In[44]:=
```
showClassifier[digitsPixJoulie[[ ;; , 3]], integerListJoulie[[ ;; , 3]]]
```

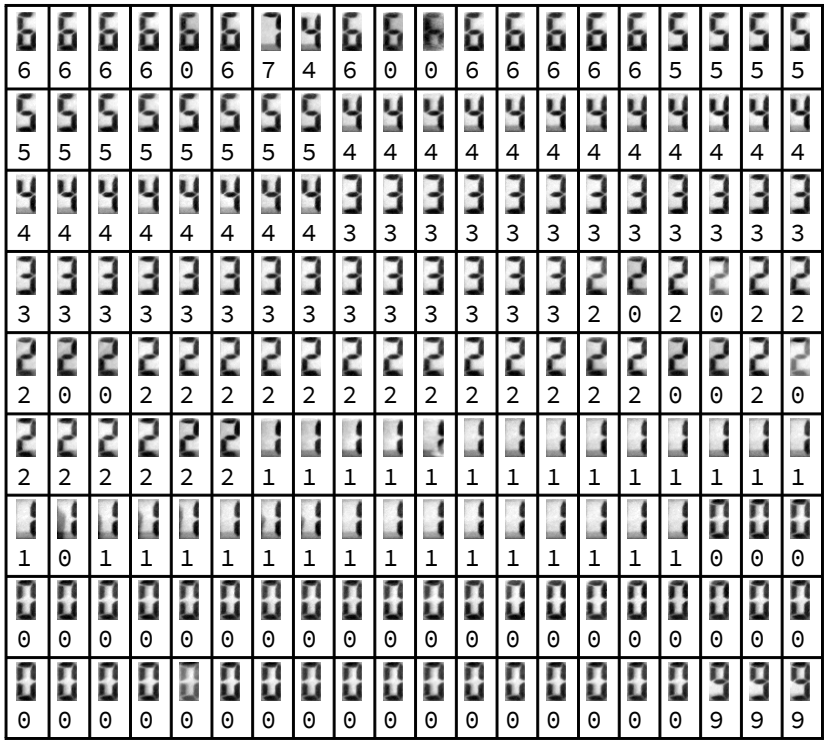
Out[44]=



Ten's place:

In[45]:=
```
showClassifier[digitsPixJoulie[[ ;; , 2]], integerListJoulie[[ ;; , 2]]]
```

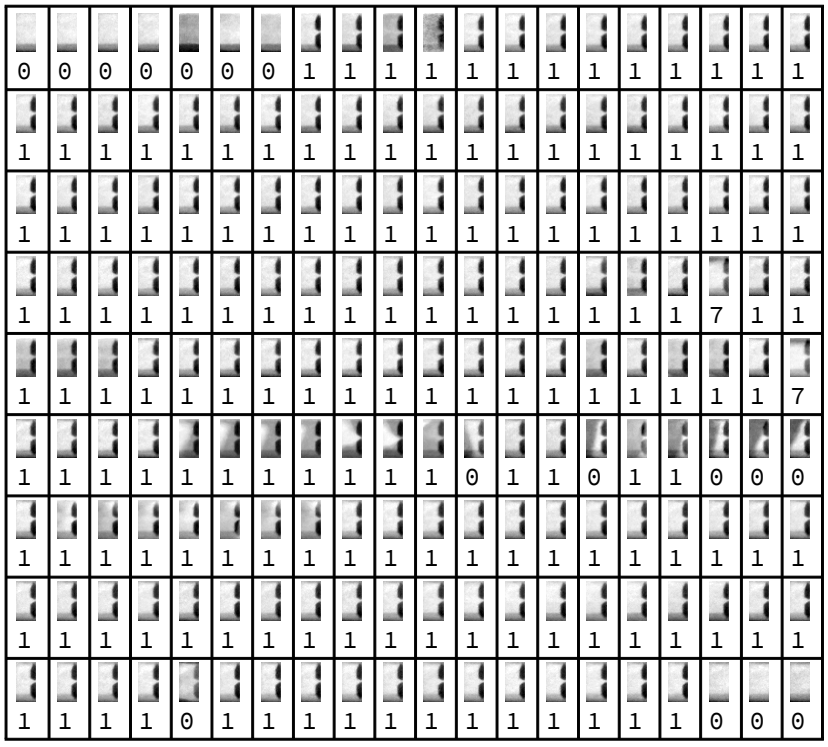
Out[45]=



Hundred's place:

In[46]:=
```
showClassifier[digitsPixJoulie[[ ;; , 1]], integerListJoulie[[ ;; , 1]]]
```

Out[46]=



Tab view:

In[47]:=
```
(*labels={"Hundred's Place","Ten's Place","One's Place"};
 TabView[Table[labels[[i]]->
    showClassifier[digitsPixJoulie[[;;,i]],integerListJoulie[[;;,i]],{i,3}],3]
*)
```

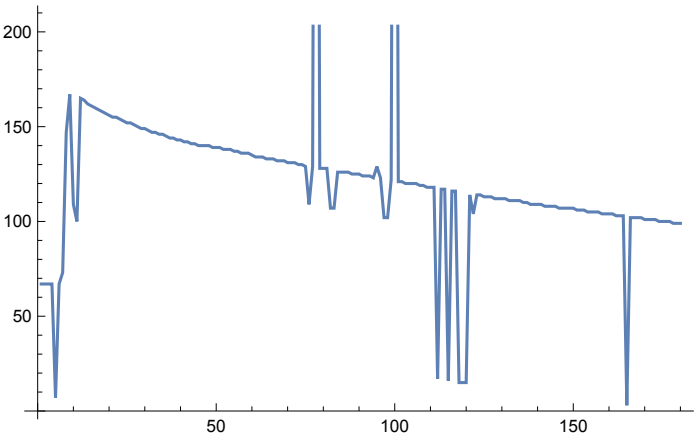
Awesome.

In[48]:=
```
joulieTemperatures = integerListJoulie // Transpose // FromDigits;
Short@joulieTemperatures
```

Out[49]//Short=
```
{67, 67, 67, 67, 7, 67, 73, 147, 167, 109, 100, 165, 164, 162, 161, 160, 159, 158, 157,
  156, 155, 155, 154, 153, 152, <<130>>, 105, 105, 104, 104, 104, 104, 103, 103,
  103, 3, 102, 102, 102, 102, 101, 101, 101, 101, 100, 100, 100, 100, 99, 99, 99}
```

In[50]:=
```
ListPlot[joulieTemperatures, Joined → True]
```

Out[50]=



Not bad.

Get Digits of Control thermometer

Repeat the above tweaking for the ‘Control’ thermometer.

Trim

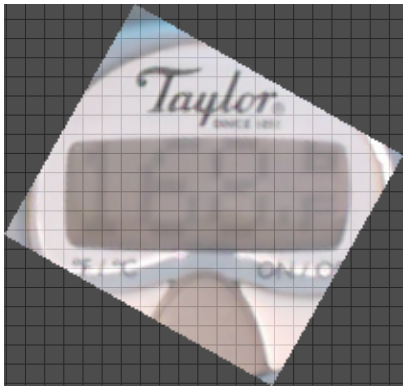
Crop (trim) the image to this bounding box around the Control therm LCD:

In[51]:=
```
Manipulate[
 trimControl = (ImageRotate[#, d Degree] &) @* (ImageTrim[#, {t1, t2}] &);
 {Show[images[[i]], Epilog → {Opacity[.2], Blue, Rectangle[t1, t2]}, ImageSize → 400],
  Show[SetAlphaChannel[trimControl@images[[i]], .7],
   GridLines → (Range[1, #, 10] & /@ ImageDimensions[trimControl@images[[i]]])],
  t1, t2
 },
 {{i, 33, "Frame"}, 1, Length@images, 1, Appearance → "Open"},
 {{t1, {498, 364}}, Locator}, {{t2, {656, 500}}, Locator},
 {{d, -29.5}, -180, 180, Appearance → "Open"}
]
```

Frame 33

d −29.5

Out[51]=



, {498, 364}, {656, 500}}

In[52]:=
```
trimControl@images[[33]]
```

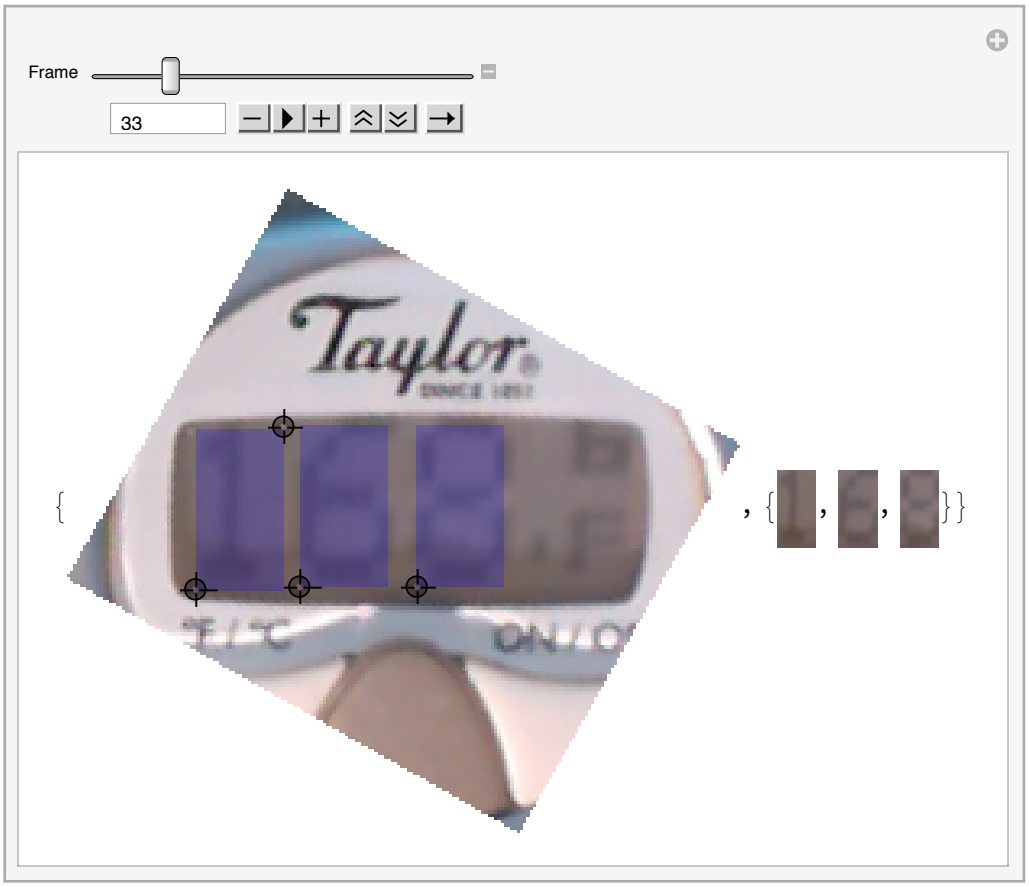
Out[52]=

Extract digits from picture

Bounding boxes around three LCD digits:

In[53]:=
```
Manipulate[Module[{im, box1, box2, box3},
  im = trimControl[images[[i]]];
  box1 = {c100a, c100b};
  box2 = {c10a, c10a + (c100b - c100a)};
  box3 = {c1a, c1a + (c100b - c100a)};
  getDigitsControl =
   Table[ImageResize[ImageTrim[#, b], {20, 40}], {b, {box1, box2, box3}}] &;
  {Show[im, Epilog → {Opacity[.2], Blue, Rectangle @@@ {box1, box2, box3}},
    ImageSize → 350], getDigitsControl[im]}
 ],
 {{i, 33, "Frame"}, 1, Length@images, 1, Appearance → "Open"},
 {{c100a, {40, 75}}, Locator}, {{c100b, {67, 125}}, Locator},
 {{c10a, {72, 76}}, Locator}, {{c1a, {108, 76}}, Locator}
]
```

Out[53]=



In[54]:=
```
getDigitsControl@trimControl@images[[33]]
```

Out[54]=

Compute all digits

In[55]:=
```
digitsControl = getDigitsControl @* trimControl /@ images;
digitsControl =
  Map[ImageAdjust @* (ColorConvert[#, "Grayscale"] &), digitsControl, {2}];
digitsControl[[
 33]]
```

Out[57]=

Classify digits as integers

In[58]:=
```
Manipulate[digitsControl[[i]], {i, 1, Length@digitsControl, 1, Appearance → "Open"}]
```

Out[58]=

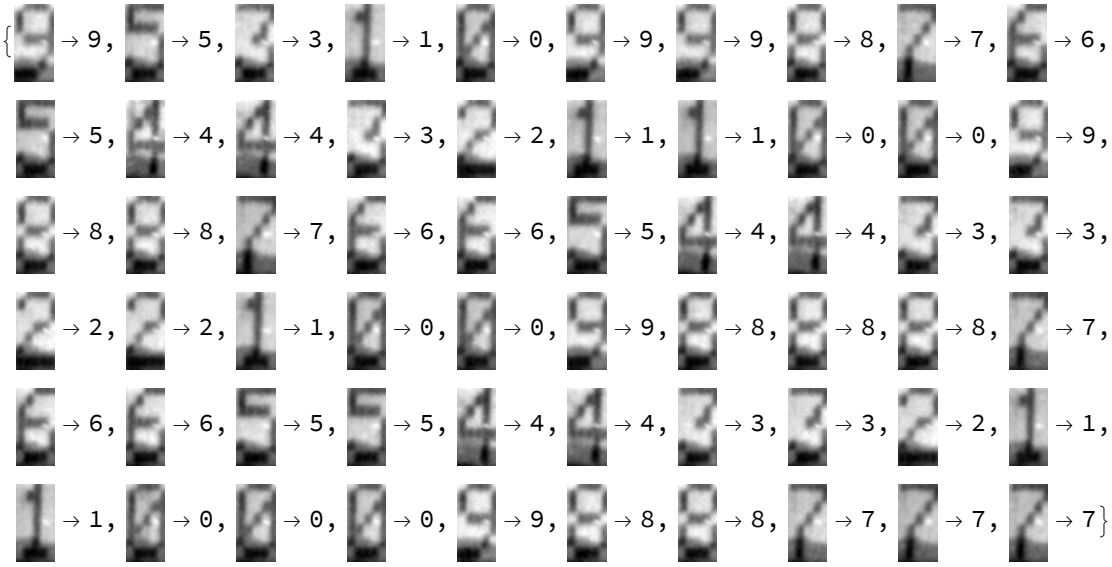
Put it on auto-play, watch it and write it down. Starting at index 13, what's the least-sig digit?

In[59]:=
```
listControl = ToExpression /@
   Characters@"9531099876544321100988766544332210098887665544332110009887777";
```

Note: Coulda just repeated this for the whole file. But if it was really long, couldn't.

In[60]:=
```
trainingDataControl = Table[
    Rule @@ {digitsControl[[12 + i, 3]], listControl[[i]]}, {i, Length@listControl}];
Short@trainingDataControl
```

Out[61]//Short=



Build classifier

In[62]:=
```
{timing, c2} = AbsoluteTiming[Classify[trainingDataControl,
    PerformanceGoal → "Quality", Method → "SupportVectorMachine"]]
{timing, integersControl} = AbsoluteTiming[Map[c2, digitsControl, {2}]];
timing
Short@integersControl
```

Out[62]=
```
{19.4149, ClassifierFunction[ Input type: Image
                              Number of classes: 10 ]}
```

Out[64]=
```
19.2325
```

Out[65]//Short=
```
{{4, 4, 7}, {4, 4, 7}, {4, 4, 7}, {4, 4, 7}, {4, 4, 7}, {4, 4, 7}, {4, 4, 7}, {4, 4, 7},
 {4, 4, 7}, {4, 4, 0}, {4, 4, 4}, {4, 4, 2}, <<157>>, {4, 4, 8}, {4, 4, 8}, {4, 4, 8},
 {4, 4, 8}, {4, 4, 7}, {4, 4, 7}, {4, 4, 7}, {4, 4, 6}, {4, 4, 6}, {4, 4, 6}, {4, 4, 6}}
```

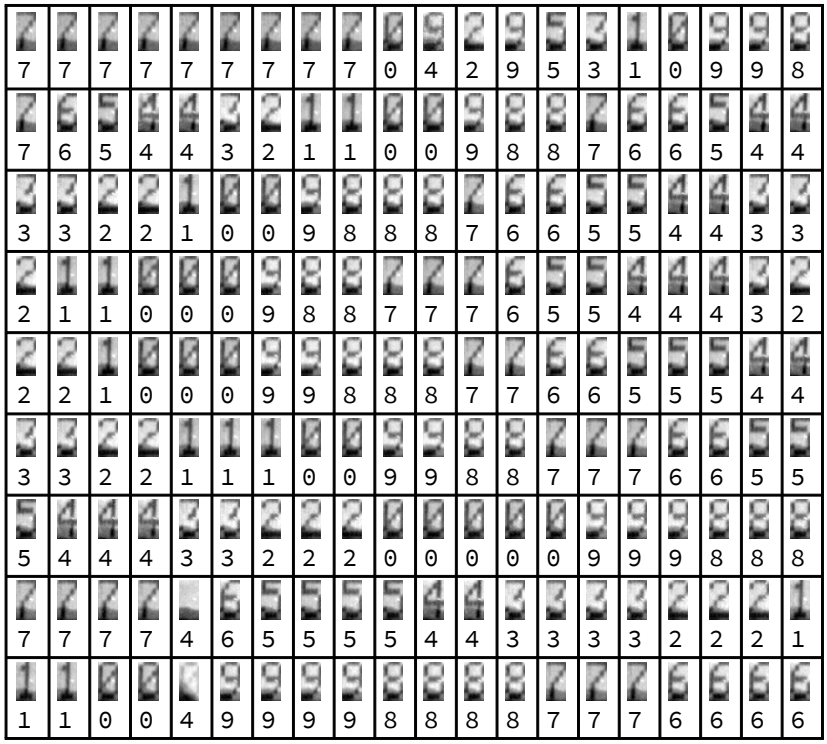
Assess classifier

How does it work? We trained it mostly on the one's place LCD segment. Perhaps it will work on the ten's and hundred's segments too?

One's digit:

In[66]:=
```
showClassifier[digitsControl[[ ;; , 3]], integersControl[[ ;; , 3]]]
```

Out[66]=

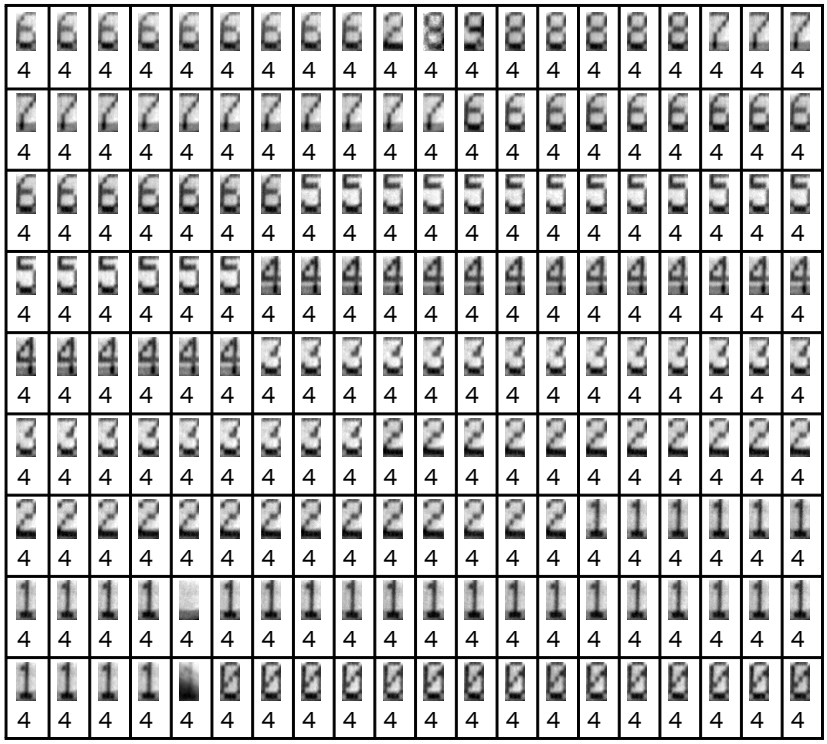


Seems pretty good!

Ten’s digit:

In[67]:=
```
showClassifier[digitsControl[[ ;; , 2]], integersControl[[ ;; , 2]]]
```

Out[67]=

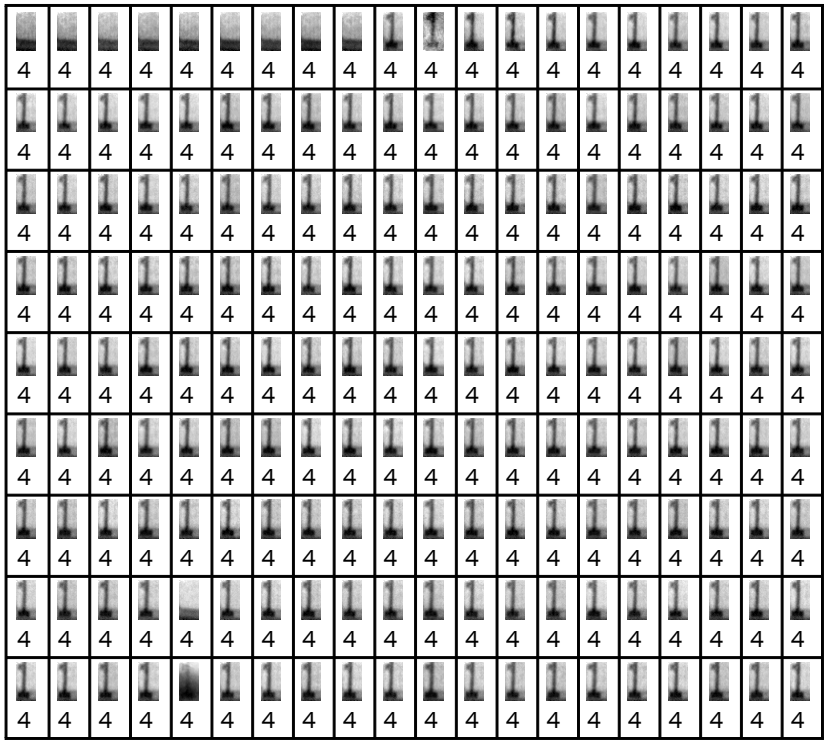


Not great.

Hundred’s digit:

In[68]:=
```
showClassifier[digitsControl[[ ;; , 1]], integersControl[[ ;; , 1]]]
```

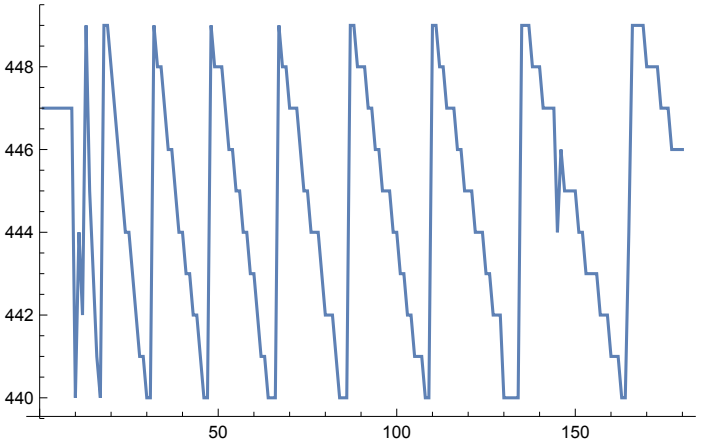
Out[68]=



In[69]:=
```
controlTemperatures = integersControl // Transpose // FromDigits;
```

In[70]:=
```
ListPlot[controlTemperatures, Joined → True]
```

Out[70]=



Since it gets so many of the tens' and hundred's wrong, let's input them manually.

Manually find the hundreds' and tens' digits

The above classification is very sensitive to brightness of the images. The classifier was trained from the one's digit, and sometimes it doesn't do very well on the ten's or hundred's digits. Since the hundred's and ten's place doesn't change very quickly, it's not much trouble to just write them down manually.

In[71]:=
```
Manipulate[digitsControl[[i]], {i, 1, Length@digitsControl, 1, Appearance → "Open"}]
```

Out[71]=

The format *x*, *y* → *z* means: frames *y* thru *z* display digit *x*.

In[72]:=
```
Clear[toRange]
toRange[{x_, y_ → z_}] := ConstantArray[x, z - y + 1];
toRange[list_] := list // Partition[#, 2] & // Map[toRange, #] & // Flatten
```

Ten's place:

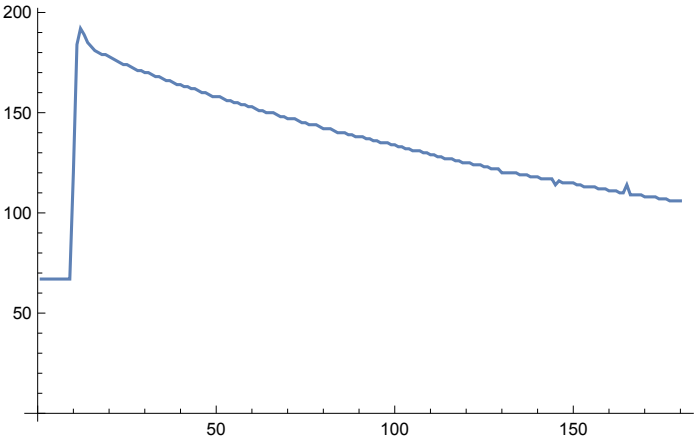
In[75]:=
```
controlTensPlace =
  toRange@{6, 1 → 9, 2, 10 → 10, 8, 11 → 11, 9, 12 → 12, 8, 13 → 17, 7, 18 → 31, 6, 32 → 47,
    5, 48 → 66, 4, 67 → 86, 3, 87 → 109, 2, 110 -> 134, 1, 135 → 165, 0, 166 → 180};
```

Hundred's place:

In[76]:=
```
controlHundredsPlace = toRange@{0, 1 → 9, 1, 10 → 180};
```

In[77]:=
```
controlOnesPlace = integersControl[[ ;; , 3]];
controlTemperatures =
  FromDigits /@ ({controlHundredsPlace, controlTensPlace, controlOnesPlace}^T);
ListPlot[controlTemperatures, Joined -> True]
```

Out[79]=



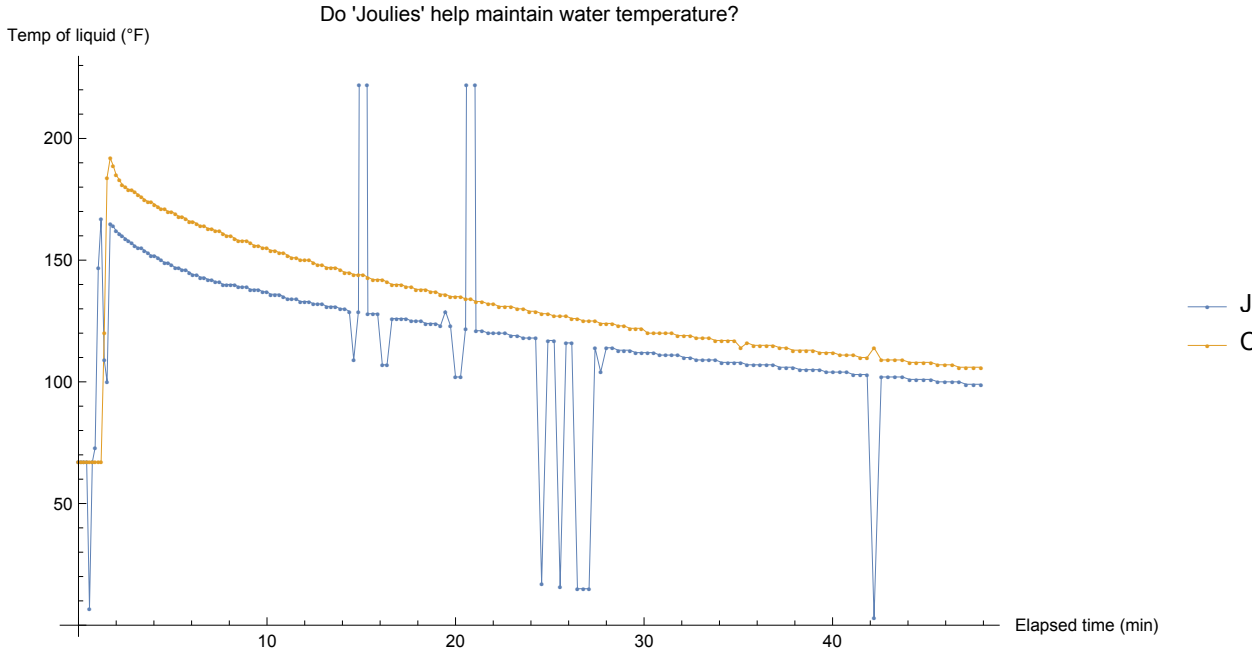
Future work: Another workaround: use the one's digit 0->9 transition to tell when the ten's digit should change

The one's digit is well-classified after it stops bonking around after index 15 or so. So let's just enter manually for 1-15 and then rely on the one's digit to tell us when it's changing:

Plot results

In[80]:=
```
plot = ListPlot[{{deltaT, joulieTemperatures}ᵀ, {deltaT, controlTemperatures}ᵀ},
  AxesLabel → {"Elapsed time (min)", "Temp of liquid (°F)"},
  PlotLegends → {"Joulie Cup", "Control Cup"},
  PlotMarkers → {●, 5}, Joined → True, PlotStyle → Thin,
  PlotLabel → "Do 'Joulies' help maintain water temperature?", ImageSize → 600]
```

Out[80]=



Import Dad's data

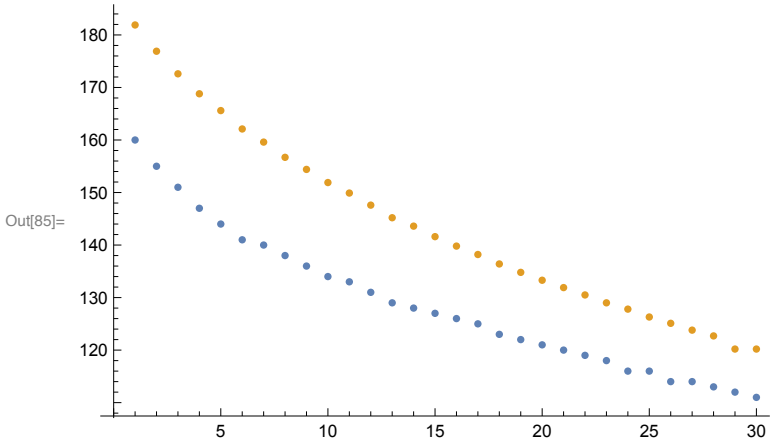
In[81]:=
```
dad = Import[
    "/Users/justin/Projects/Joulies/Analysis/DadsCamera_Analysis/data_dadcamera.txt
      ", "Table"];
Length@dad
```

Out[82]=
```
30
```

In[83]:=
```
dadJoulie = First@Transpose@dad;
dadControl = Last@Transpose@dad;
```

In[85]:=
```
ListPlot[{dadJoulie, dadControl}]
```

Out[85]=



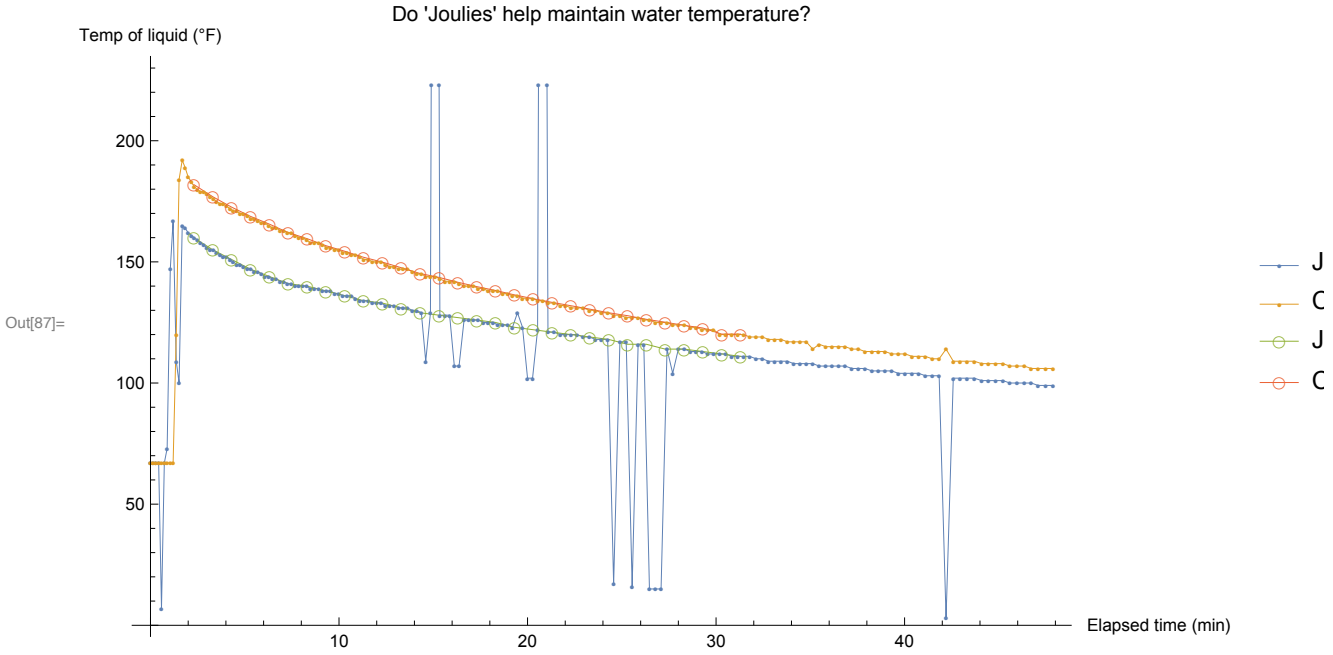
I shifted dad's time axes to get it to line up:

In[86]:=
```
dadTime = Range[Length@dadJoulie] + 1.3;
```

Plot all data

In[87]:=
```
plot = ListPlot[{{deltaT, joulieTemperatures}ᵀ,
    {deltaT, controlTemperatures}ᵀ, {dadTime, dadJoulie}ᵀ, {dadTime, dadControl}ᵀ},
  AxesLabel → {"Elapsed time (min)", "Temp of liquid (°F)"},
  PlotLegends → {"Joulie Cup - MacBook", "Control Cup - MacBook",
    "Joulie Cup - Dad's Camera", "Control Cup - Dad's Camera"},
  PlotMarkers → {{•, 5}, {•, 5}, {○, 10}, {○, 10}}, Joined → True, PlotStyle → Thin,
  PlotLabel → "Do 'Joulies' help maintain water temperature?", ImageSize → 600]
```

Out[87]=



In[88]:=
```
(*["~/Desktop/joulie_plot1.png",plot,ImageResize→120]*)
```

In[89]:=
```
(*Export["~/Desktop/macbook_temp_data.txt",
 {QuantityMagnitude@deltaT,joulieTemperatures,controlTemperatures}^T,"Table"]*)
```

Future work: Calculate efficiency

JPP: I think we still need to know how much water was in each of the cups. . . . Dad can measure the volume of his mugs.

Dad: Up to the water level visible in the Joulie-filled cup is 276 ml, give or take about 5%.

Spence: The Joulie information packet says: “0.75oz/1.5Tbsp displaced per Joulie; 4oz/118mL regulated per Joulie.”

In[90]:=
```
numJoulies = 5;
fullCupMl = 276; (* ml *)
mlPerTbsp = 14.7868; (* source: google *)
joulieVolTbsp = 1.5;
totalJoulieVolMl = numJoulies * joulieVolTbsp * mlPerTbsp;
volWaterJoulie = fullCupMl - totalJoulieVolMl
```

Out[95]=
```
165.099
```

Specific heat of water:

TODO: more on this.