Reading an analog dial with image processing

Justin Pearson
2014-10-19

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

Function to display video metadata

In[3]:=
```
info[f_] :=
 Module[{elems = {"BitDepth", "ColorSpace", "Duration", "FrameCount", "FrameRate",
     "ImageSize", "VideoEncoding"}, i, e},
  Print["Filesize: ", FileByteCount[f], " bytes"];
  For[i = 1, i ≤ Length@elems, i++,
   e = elems[[i]];
   Print[e, " : ", Import[f, e]];
  ]
 ]
```

In[4]:=
```
oneframe[f_] := Import[f, {"ImageList", 100}]
```

In[5]:=
```
show[f_] := (info[f]; oneframe[f])
```

Video of gas meter

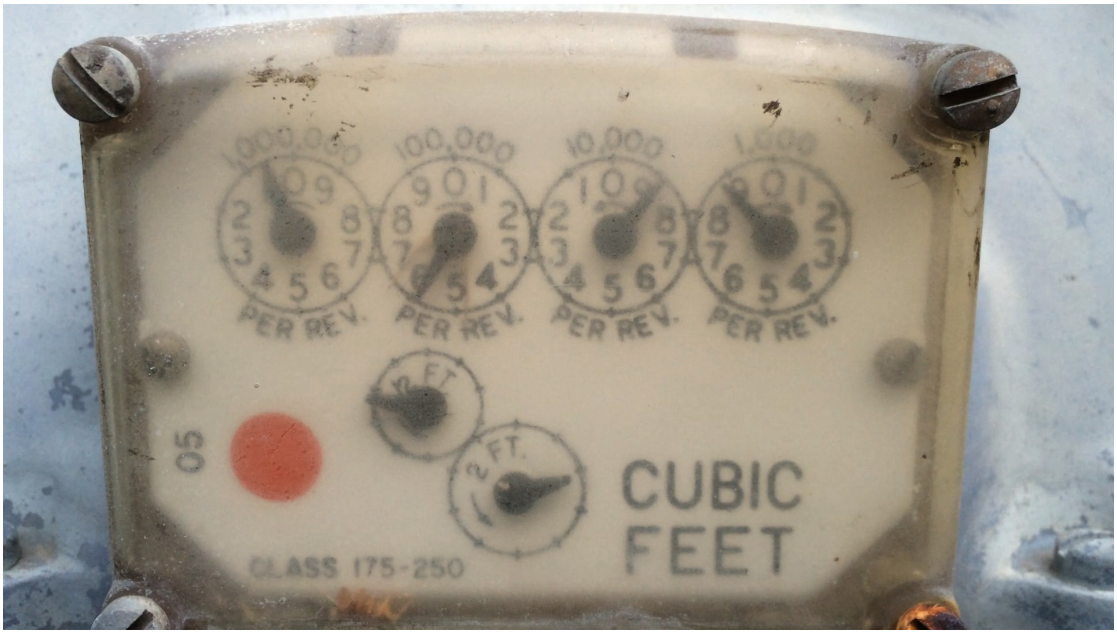
I recorded a video of the gas line next to my apartment:

In[6]:=
```
fname = "IMG_1310.MOV";
```

In[7]:=
```
show@fname
```

```
Filesize: 119 617 161 bytes
BitDepth : 8
ColorSpace : RGB
Duration : 61.4
FrameCount : 1842
FrameRate : 30.
ImageSize : {1920, 1080}
VideoEncoding : H.264
```

Out[7]=



Too big to import every frame of. Instead I used After Effects 6.5 to stabilize and crop it, making “dial.mov”.

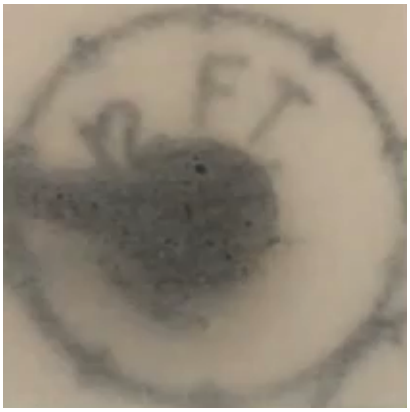
Import

In[8]:=

```
fname = "dial.mov";
show@fname
```

```
Filesize: 5985470 bytes
BitDepth : 8
ColorSpace : RGB
Duration : 60.
FrameCount : 1800
FrameRate : 30.
ImageSize : {210, 210}
VideoEncoding : H.264
```

Out[9]=



In[10]:=
```
{time, frames} = AbsoluteTiming@Import[fname, "ImageList"];
time
```

Out[11]= 14.303

In[12]:=
```
(*Export["a_frame.jpg",frames[[500]];*)
```

Manually record dial positions

In[13]:=
```
Manipulate[frames[[i]],
 {{i, 1, "frame"}, 1, Length@frames, 1, Appearance → "Open", ImageSize → 300}]
```

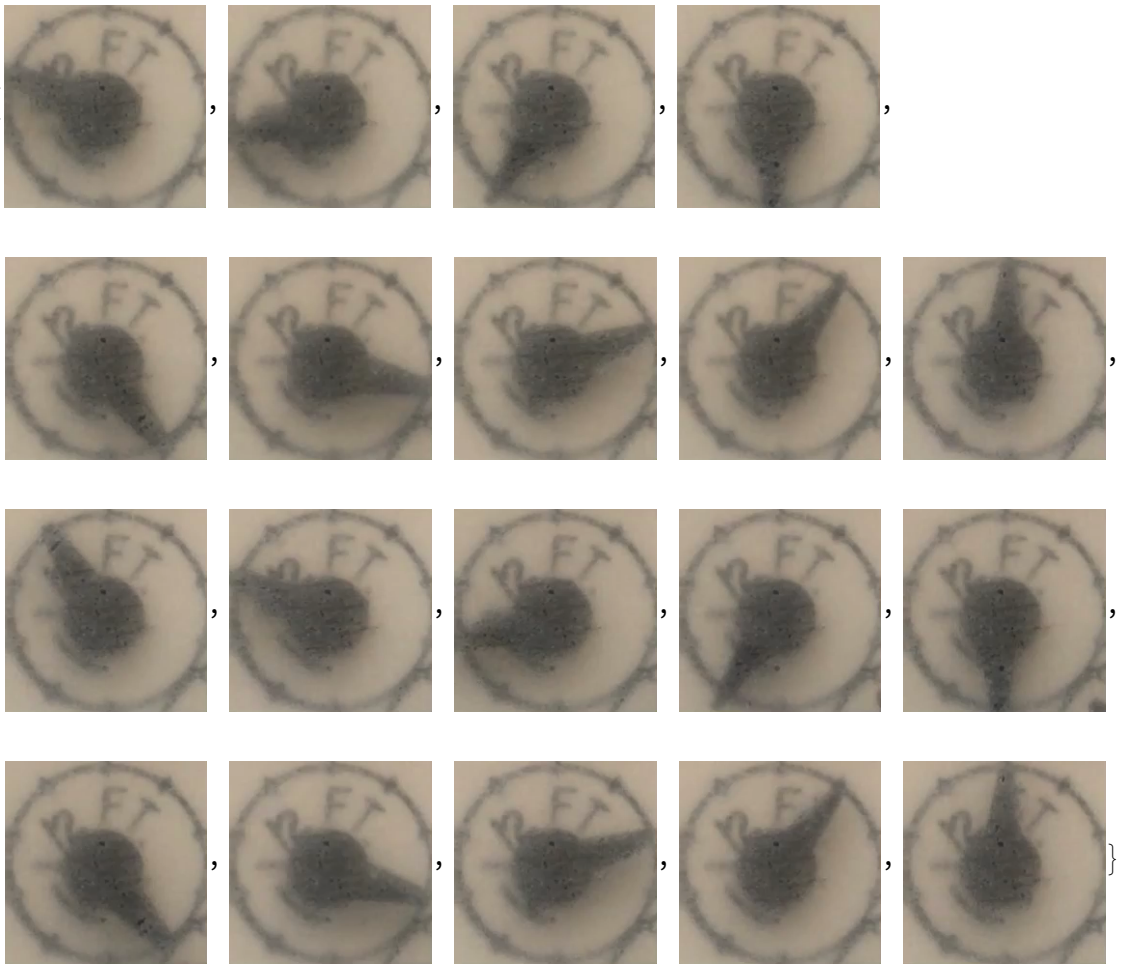
Out[13]=

Manually record roughly what frames show which dial positions. Despite this dial rotating CCW, we label the positions like clock hands, clockwise with 0 at the top, up to 9.

In[14]:=
```
frameind2clock = {47 → 8, 160 → 7, 265 → 6, 370 → 5,
    469 → 4, 589 → 3, 668 → 2, 743 → 1, 842 → 0, 923 → 9, 1008 → 8, 1108 → 7,
    1203 → 6, 1310 → 5, 1408 → 4, 1503 → 3, 1584 → 2, 1657 → 1, 1753 → 0};
traininginds = frameind2clock[[ ;; , 1]]
trainingpix = frames[[traininginds]]
trainingclockpositions = frameind2clock[[ ;; , 2]] // N
```

Out[15]=
```
{47, 160, 265, 370, 469, 589, 668, 743, 842,
 923, 1008, 1108, 1203, 1310, 1408, 1503, 1584, 1657, 1753}
```

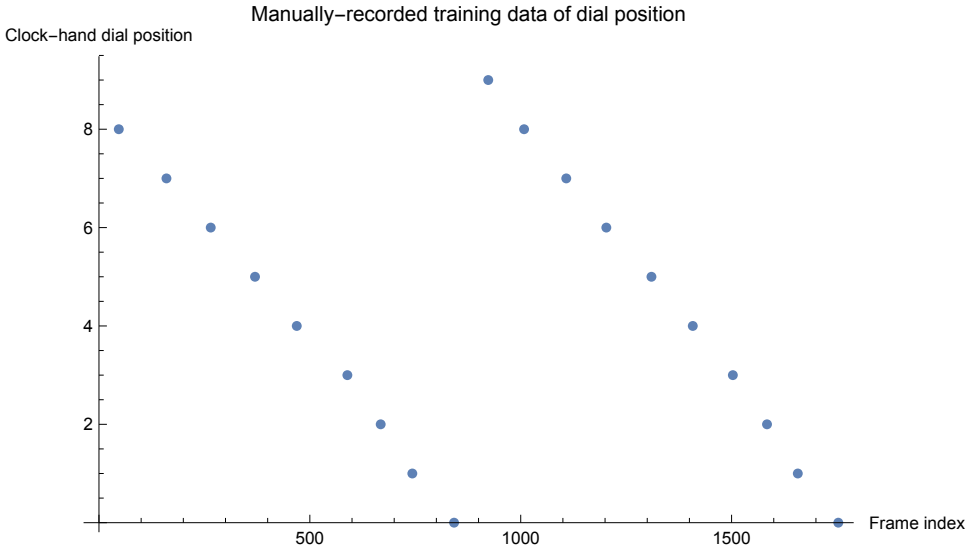
Out[16]=



Out[17]=
```
{8., 7., 6., 5., 4., 3., 2., 1., 0., 9., 8., 7., 6., 5., 4., 3., 2., 1., 0.}
```

In[18]:=
```
ListPlot[{traininginds, trainingclockpositions}^T,
 AxesLabel → {"Frame index", "Clock-hand dial position"},
 PlotLabel → "Manually-recorded training data of dial position", ImageSize → 500]
```

Out[18]=



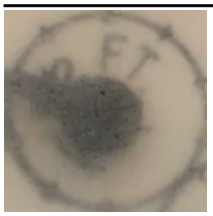
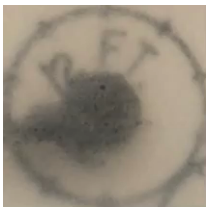
That discontinuity where it wraps will cause problems, as we'll see.

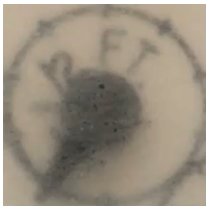
It'll be nice to draw a cute picture of a gauge.

In[19]:=
```
gauge[pos_] := AngularGauge[pos, {0, 10},
  ScaleOrigin → {π/2, Pi/2 - 2 Pi}, ScaleDivisions → 10, ImageSize → 100]
```

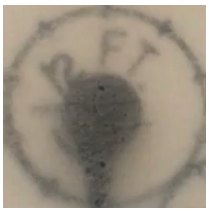
In[20]:=
```
MapThread[{#1, #2, gauge@#2} &, {trainingpix, trainingclockpositions}] //
 TableForm[#,
   TableAlignments → Center,
   TableHeadings → {None,
     {"training pic", "dial position\n(read manually)", "pretty position"}}
  ] &
```

Out[20]//TableForm=

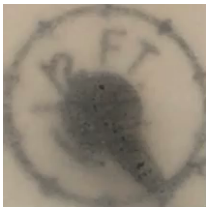
training pic	dial position (read manually)	pretty position
	8.	
	7.	



6.



5.

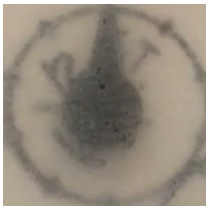


4.

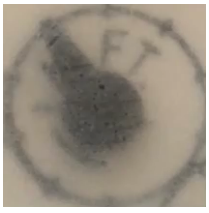
3.

2.

1.

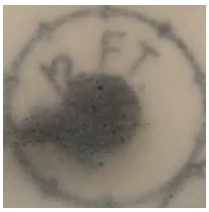


0.

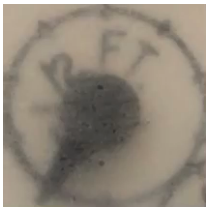


9.

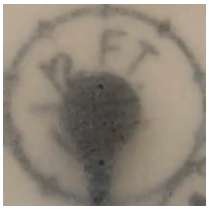
8.



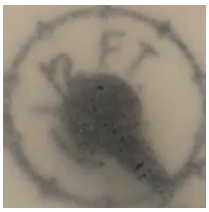
7.



6.

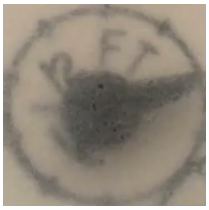


5.



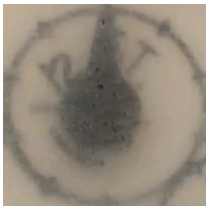
4.

3.



2.

1.



0.

Simplest place to start: use built-in Predict[]

There are a lot of built-in methods that Predict[] uses:

In[21]:=
```
methods = {"LinearRegression", "NearestNeighbors",
   "NeuralNetwork", "RandomForest", "GaussianProcess"};
```

Will any of them work out-of-the-box?

In[22]:=
```
trainingdata = trainingpix → trainingclockpositions;
```

In[23]:=
```
{time, predictors} = AbsoluteTiming@Table[
    Predict[trainingdata, Method → m, PerformanceGoal → "Quality"], {m, methods}];
time
```

Out[24]= 17.616

Weirdly, the LinearRegression predictor varies across subsequent invocations:

In[25]:=
```
Do[
 Print@PredictorInformation@Predict[trainingpix → trainingclockpositions,
    Method → "LinearRegression", PerformanceGoal → "Quality"],
 2]
```

Predictor information

Method	Linear regression
Number of features	1
Number of training examples	19
L1 regularization coefficient	0
L2 regularization coefficient	0.00001

Predictor information

Method	Linear regression
Number of features	1
Number of training examples	19
L1 regularization coefficient	0
L2 regularization coefficient	10000.

In[26]:=
```
Grid[{methods, PredictorInformation /@ predictors}^T, Frame → All]
```

Out[26]=

LinearRegression	Predictor information Method: Linear regression Number of features: 1 Number of training examples: 19 L1 regularization coefficient: 0 L2 regularization coefficient: 0.00001
NearestNeighbors	Predictor information Method: K−nearest neighbors Number of features: 1 Number of training examples: 19 Number of neighbors: 1 Distance function: EuclideanDistance
NeuralNetwork	Predictor information Method: Neural network Number of features: 1 Number of training examples: 19 L1 regularization coefficient: 0 L2 regularization coefficient: 0.1 Number of hidden layers: 1 Hidden nodes: 10 Hidden layer activation functions: Tanh CostFunction: Cost Function
RandomForest	Predictor information Method: Random forest Number of features: 1 Number of training examples: 19 Number of trees: 200

GaussianProcess	Predictor information **Method** Gaussian Process **Number of features** 1 **Number of training examples** 19 **AssumeDeterministic** False **Numerical Covariance Type** SquaredExponential **Nominal Covariance Type** HammingDistance **EstimationMethod** MaximumPosterior **OptimizationMethod** Automatic

Does any of these predictors work out-of-the-box?

Unsurprisingly, the predictors are decent predicting the data they were trained on:

In[27]:=
```
{time, guessesOnTraining} =
  AbsoluteTiming@Table[p@f, {f, trainingpix}, {p, predictors}];
time
```

Out[28]= 14.9416

In[29]:=
```
Short[guessesOnTraining]
```

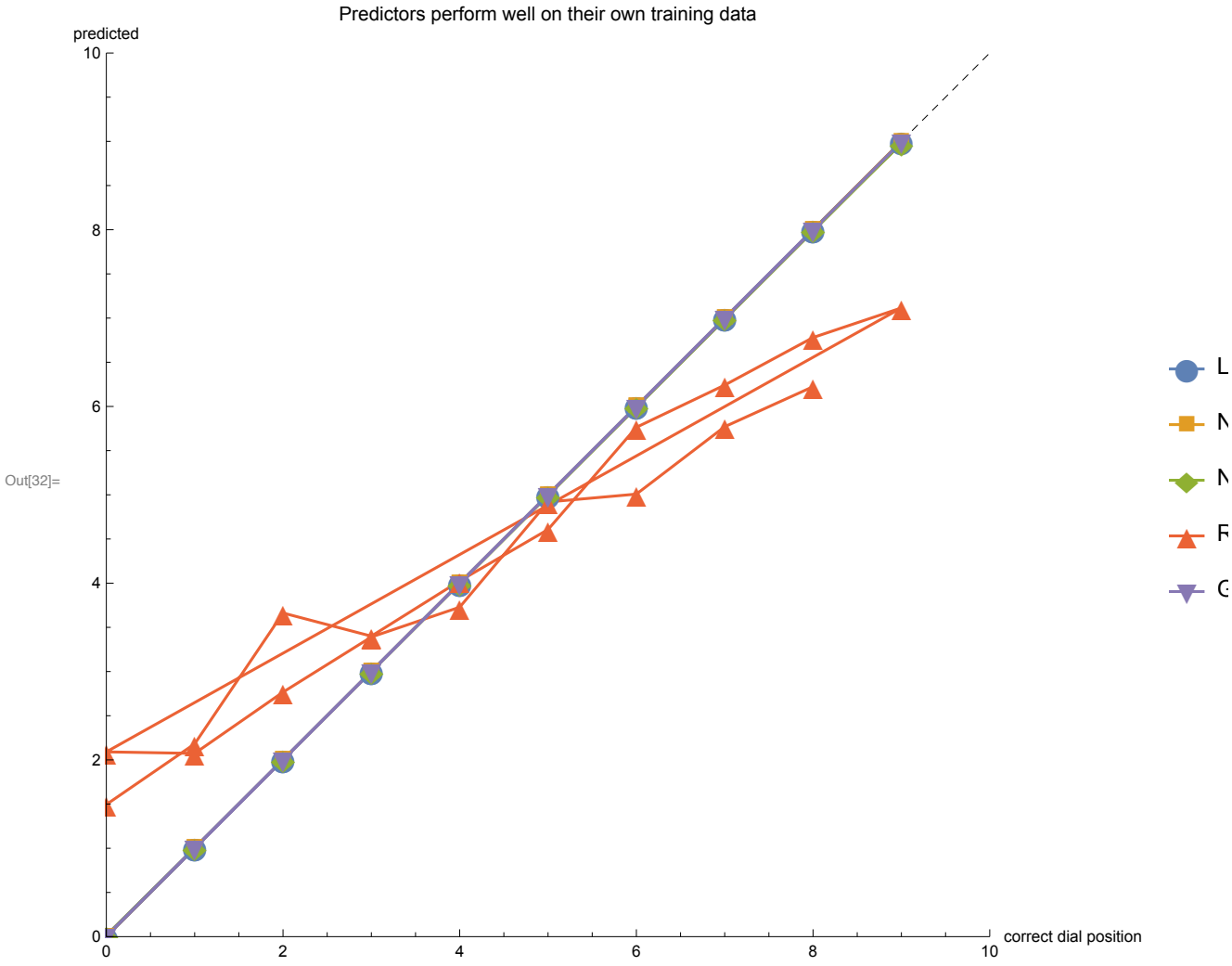
Out[29]//Short= $\{\{8., 8., 7.99835, 6.22375, 7.99588\}, \ll 17 \gg ,$
$\{2.24377 \times 10^{-6}, 0., 0.0165258, 1.495, 0.00470211\}\}$

In[30]:=
```
comparePts = Table[{trainingclockpositions, g}^T, {g, guessesOnTraining^T}];
Short[comparePts]
```

Out[31]//Short= $\{\{\{8., 8.\}, \{7., 7.\}, \{6., 6.\}, \{5., 5.\}, \ll 12 \gg ,$
$\{2., 2.\}, \{1., 1.\}, \{0., 2.24377 \times 10^{-6}\}\}, \ll 3 \gg , \{\ll 1 \gg\}\}$

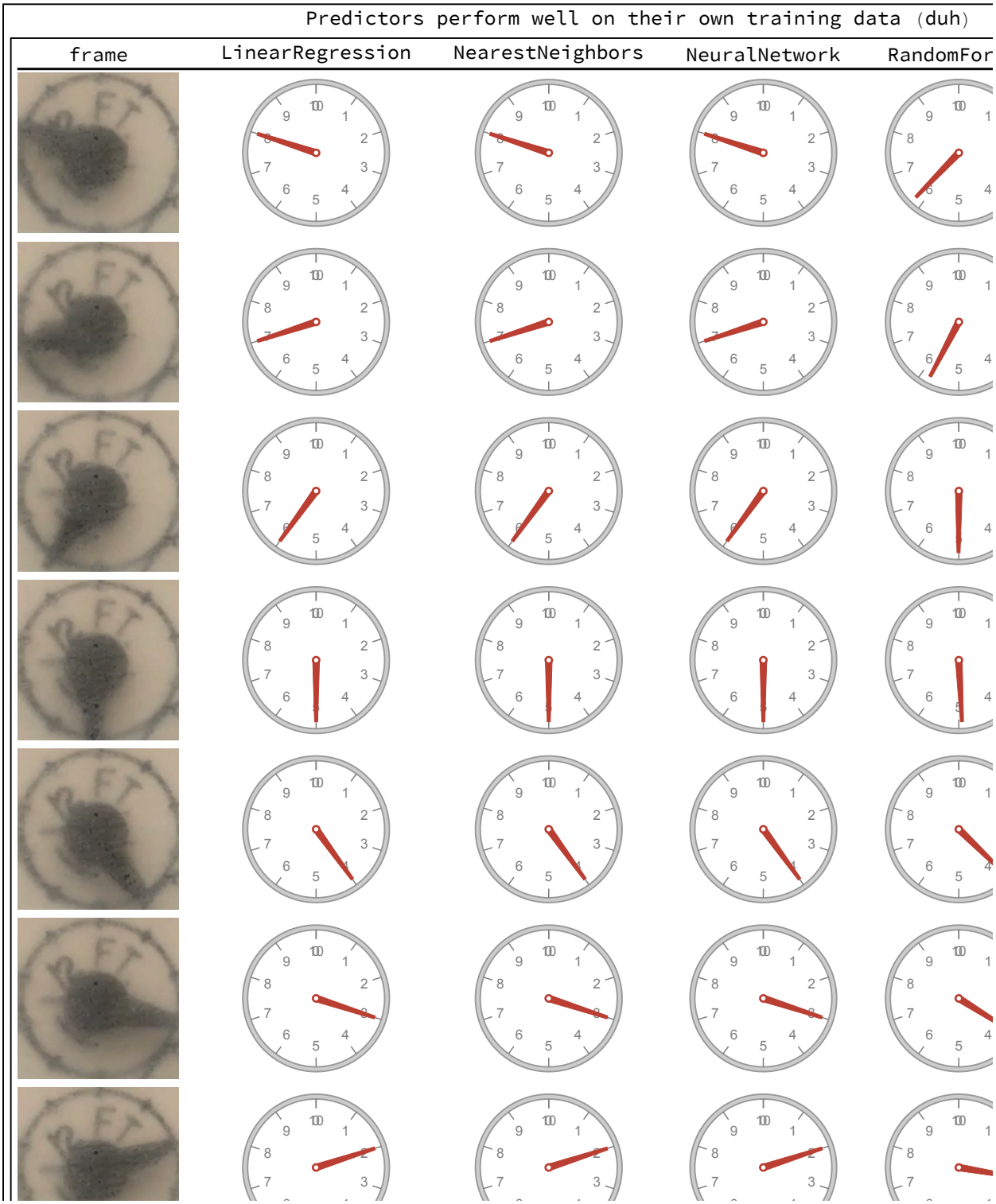
In[32]:=
```
ListLinePlot[comparePts,
 PlotLegends → methods,
 PlotMarkers → Table[{m, 20}, {m, {"●", "■", "◆", "▲", "▼"}}],
 Prolog → {Dashed, Line[{{0, 0}, {10, 10}}]},
 PlotRange → {{0, 10}, {0, 10}},
 AspectRatio → 1,
 ImageSize → 600,
 AxesLabel → {"correct dial position", "predicted"},
 PlotLabel → "Predictors perform well on their own training data"
]
```

Out[32]=

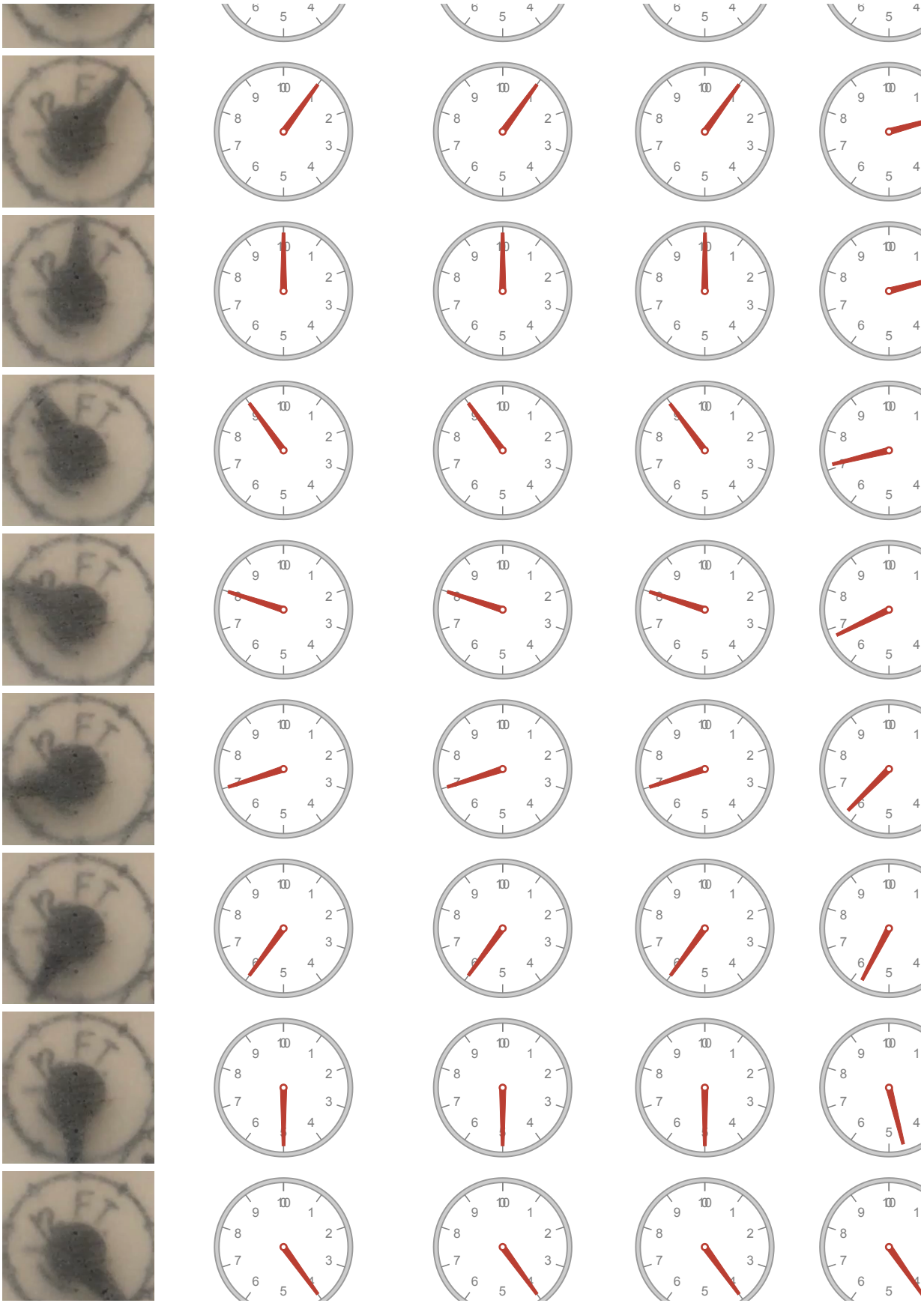


In[33]:=
```
gauges = Map[gauge, guessesOnTraining, {2}];
```

In[34]:=
```
TableForm[
  {trainingpix, gauges} // Transpose // Map[Flatten],
  TableHeadings → {None, {"frame"} ~Join~ methods},
  TableAlignments → Center] // Framed // Labeled[#,
  "Predictors perform well on their own training data (duh)", Top] & // Framed
```



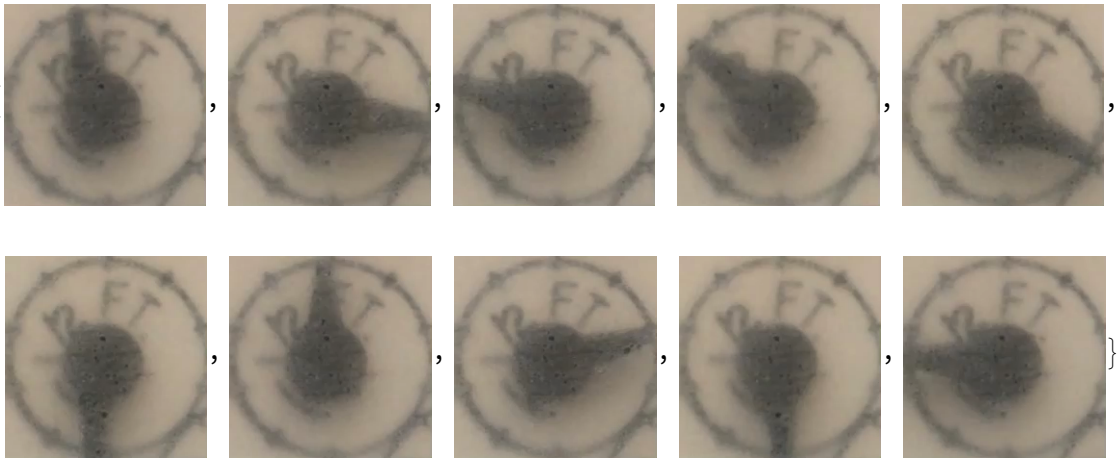
Out[34]=



But the predictors work horribly on non-training images:

In[35]:=
```
SeedRandom[1234]; testframes = RandomSample[frames, 10]
```

Out[35]=



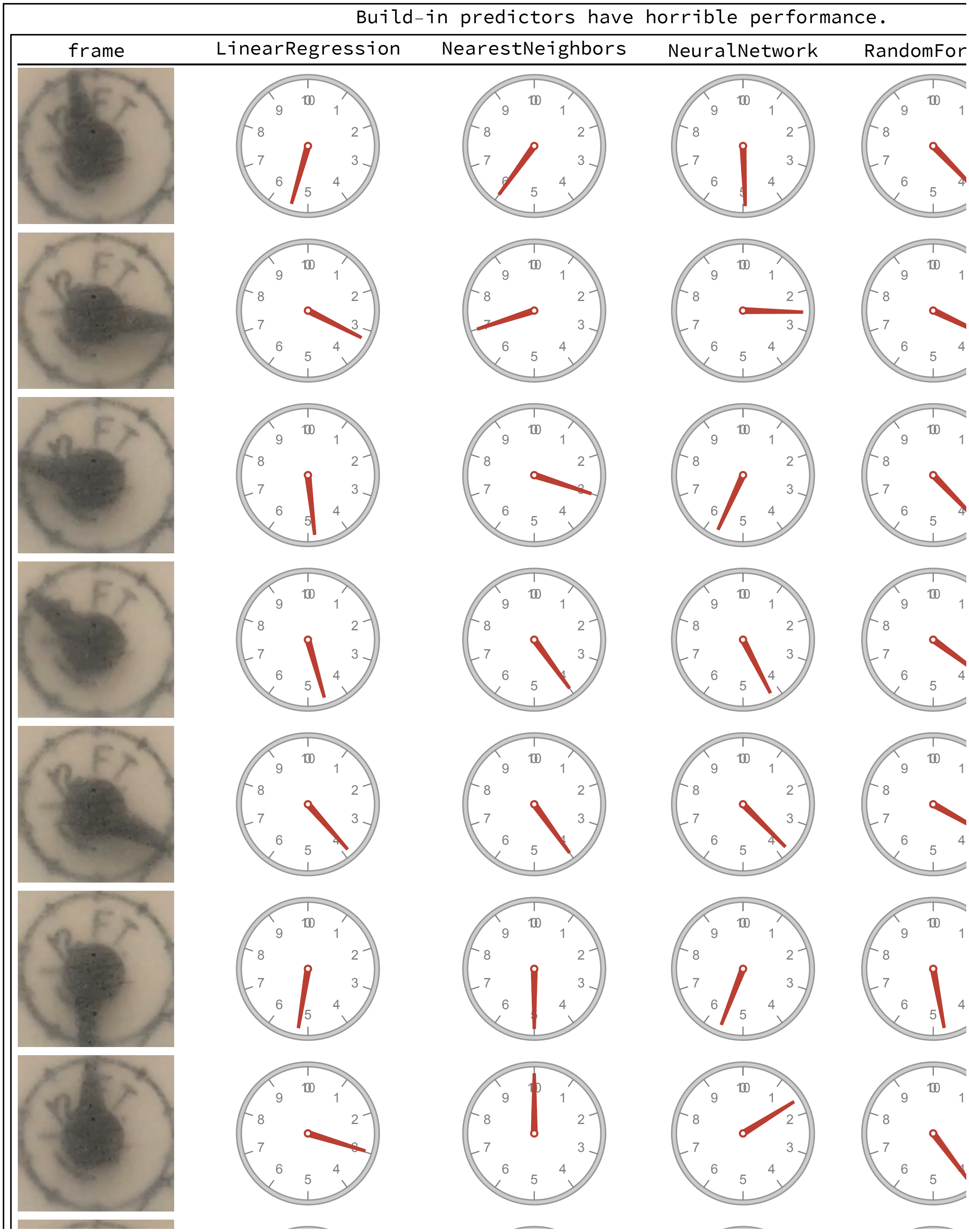
In[36]:=
```
{time, guesses} = AbsoluteTiming@Table[p@f, {f, testframes}, {p, predictors}];
time
```

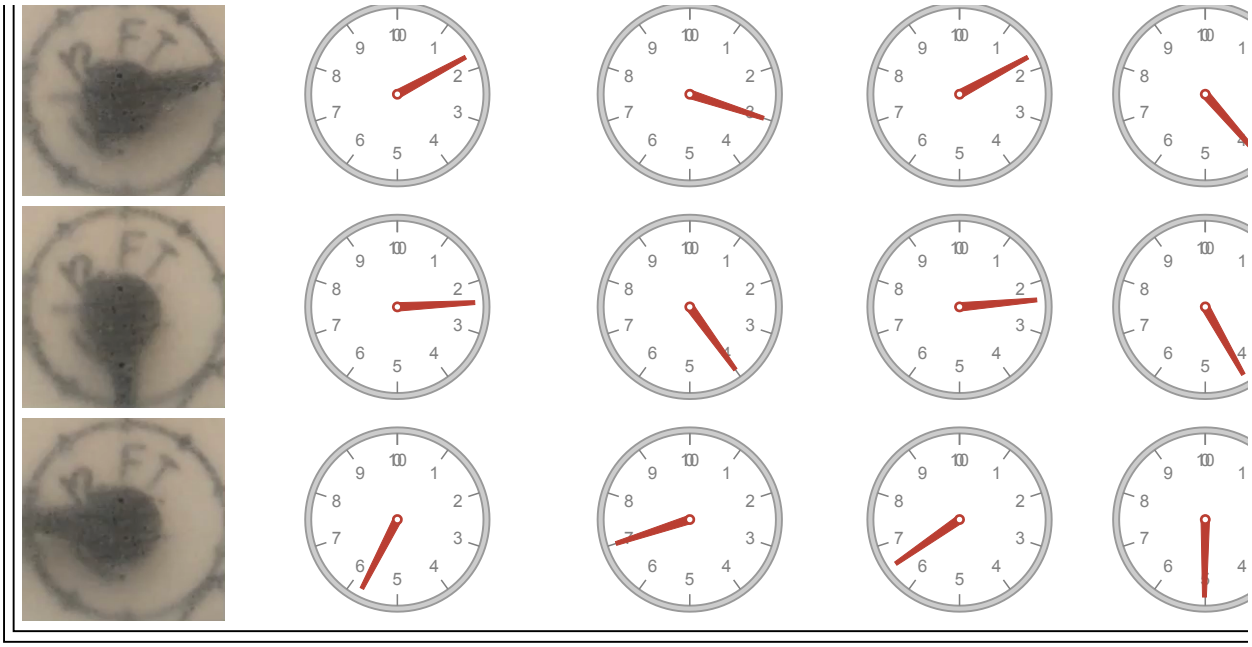
Out[37]=
```
7.94312
```

In[38]:=
```
tab = MapThread[Flatten[{{#1}, gauge /@ #2}] &, {testframes, guesses}];
```

In[39]:=
```
TableForm[tab, TableHeadings → {None, {"frame"} ~ Join ~ methods},
    TableAlignments → Center] // Framed //
  Labeled[#, "Build-in predictors have horrible performance.", Top] & // Framed
```

Out[39]=





They're all horrible. The best one is NearestNeighbors, which still sucks becuase it only returns values that were exactly in the training set (no interpolation).

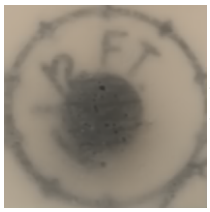
So: let's explore another technique for reading a picture of an analog dial.

Read dial with image processing

Let's see if we can do better by just using good old image processing.

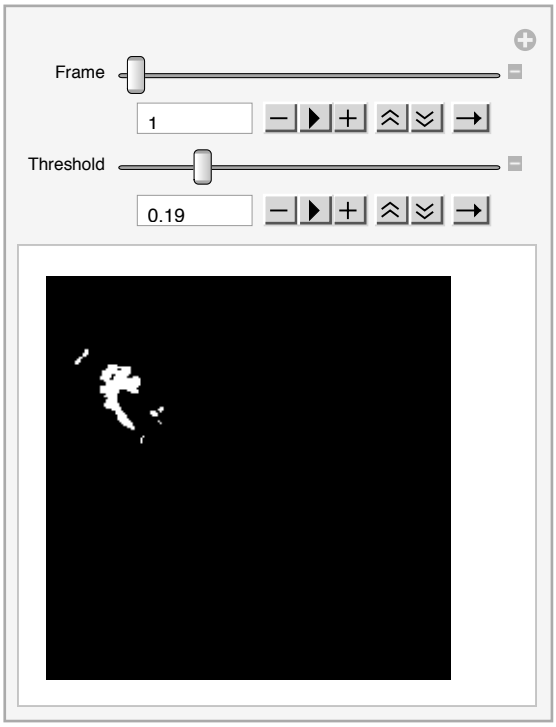
The first step in cleaning up the image feature-space is to use simpler images. Remove the background.

In[40]:=
```
{time, background} = AbsoluteTiming[frames // Map[ImageData] // Mean // Image]
```

Out[40]=
```
{11.4153,
```

```
}
```

In[41]:=
```
Manipulate[ImageDifference[frames[[i]], background] // Binarize[#, t] &,
 {{i, 1, "Frame"}, 1, Length@frames, 1, Appearance → "Open"},
 {{t, .19, "Threshold"}, 0, 1, Appearance → "Open"}
]
```

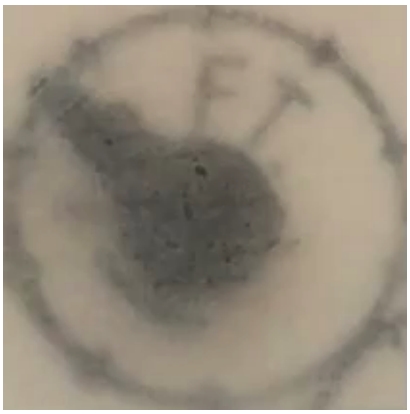
Out[41]=



Let’s dilate the image a bit, then take the centroid of the largest blob.

In[42]:=
```
im = First@frames
```

Out[42]=

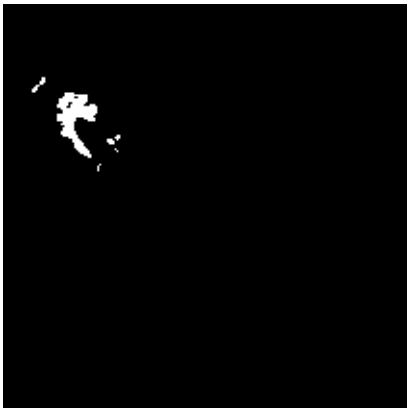


In[43]:=
```
ImageDifference[im, background]
```

Out[43]=

In[44]:=
```
ImageDifference[im, background] // Binarize[#, .19] &
```

Out[44]=



In[45]:=
```
ImageDifference[im, background] // Binarize[#, .19] & // Dilation[#, 6] &
```

Out[45]=

In[46]:=
```
im2 = %;
```

Biggest blob:

In[47]:=
```
ComponentMeasurements[im2, {"Count", "Centroid"}]
```

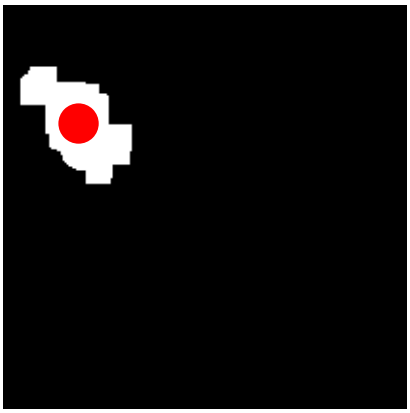
Out[47]=
```
{1 → {1984, {39.3095, 148.336}}}
```

In[48]:=
```
center = ComponentMeasurements[im2, {"Count", "Centroid"}] //
    MaximalBy[#, #[[2, 1]] &, 1] & // #[[1, 2, 2]] &
```

Out[48]= {39.3095, 148.336}

In[49]:=
```
HighlightImage[im2, Style[center, PointSize[.1]]]
```

Out[49]=

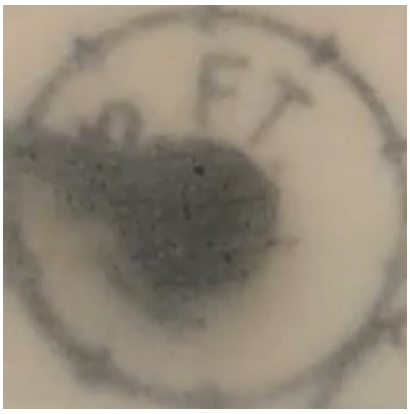


Roll it up:

In[50]:=
```
im2bw[im_] :=
  (im // ImageDifference[#, background] & // Binarize[#, .19] & // Dilation[#, 6] &)
bw2center[bw_] := (bw // ComponentMeasurements[#, {"Count", "Centroid"}] & //
     MaximalBy[#, #[[2, 1]] &, 1] & // #[[1, 2, 2]] &)
im2center = (# // im2bw // bw2center) &;
hilite[im_, center_] := HighlightImage[im, Style[center, PointSize[.1]]]
im2hilited[im_] := hilite[im, im2center@im]
im2hilitedBw[im_] := hilite[im2bw@im, im2center@im]
```

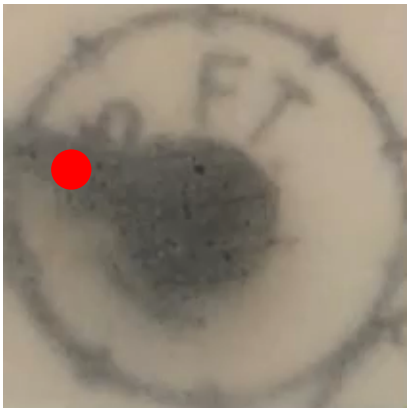
In[56]:=
```
trainingpix[[1]]
```

Out[56]=



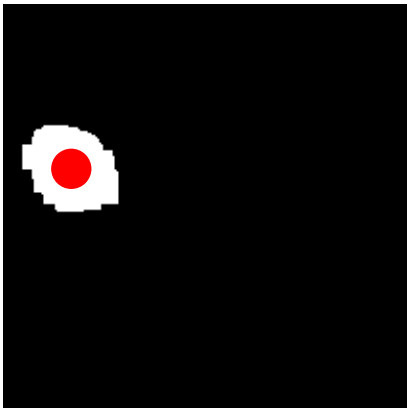
In[57]:=
```
im2hilited[trainingpix[[1]]]
```

Out[57]=



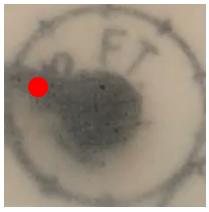
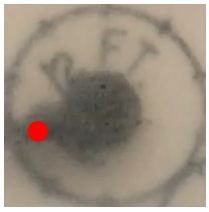
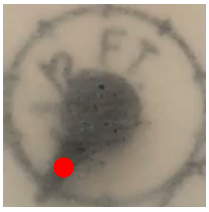
In[58]:=
```
im2hilitedBw[trainingpix[[1]]]
```

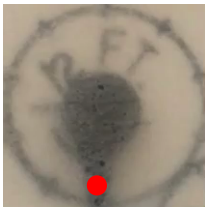
Out[58]=

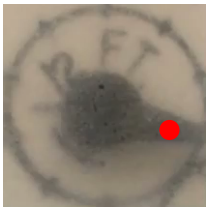
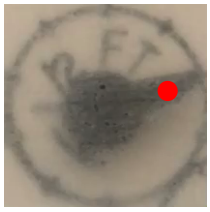


In[59]:=
```
Labeled[HighlightImage[#, Style[im2center@#, PointSize[.1]]], im2center@#] & /@
 trainingpix
```

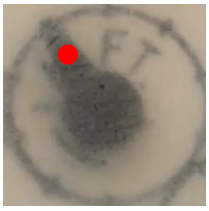
Out[59]=


{35.5253, 124.199}

{34.5694, 78.2224}

{62.3328, 40.5482}

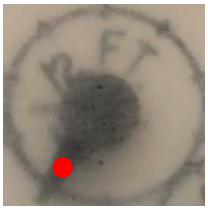
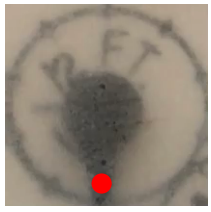
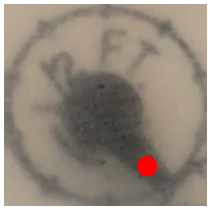

{97.7663, 21.9917}

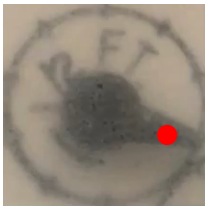
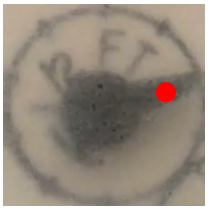
{144.117, 37.4679}

{172.697, 79.7244}

{169.45, 120.388}

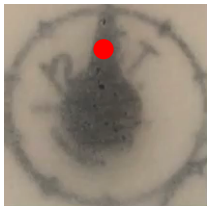

{144.681, 149.543}

{105.775, 163.899}

{66.9071, 157.356}

{36.7875, 124.412}


{34.7043, 78.0387}

{61.705, 40.2616}

{99.9716, 23.2201}

{147.756, 41.4735}


{170.898, 73.9937}

{169.189, 118.723}

{145.774, 149.314}

{102.675, 163.285}

That component-measurement thing might be good enough that we can just arctan it to get the dial position and be done with it.

In[60]:=
```
{w, h} = ImageDimensions[First@trainingpix]
```

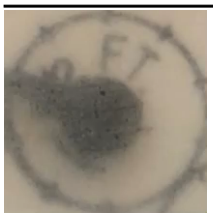
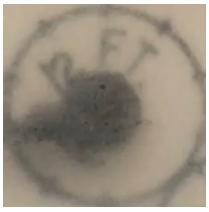
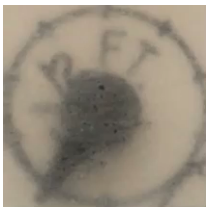
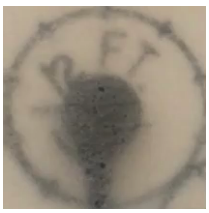
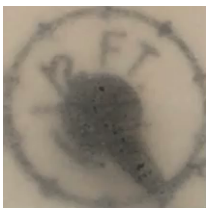
Out[60]=
```
{210, 210}
```

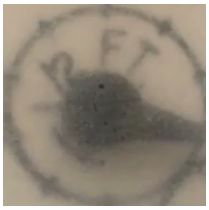
In[61]:=
```
coordToDial[coord_] :=
 (90 - (ArcTan @@ (coord - {w, h} / 2)) / Degree) / 360 * 10 // Mod[#, 10] &
```

In[62]:=
```
predict = coordToDial@*im2center;
```

In[63]:=
```
MapThread[{#1, #2, predict@#1, gauge@predict@#1} &,
  {trainingpix, trainingclockpositions}] //
 TableForm[#,
   TableAlignments → Center,
   TableHeadings → {None,
     {"training pic",
      "dial position\n(read manually)", "predicted position", "pretty"}}
  ] &
```

Out[63]//TableForm=

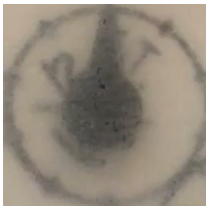
training pic	dial position (read manually)	predicted position	pretty
	8.	7.92911	
	7.	6.92176	
	6.	5.93068	
	5.	5.13835	
	4.	4.16442	



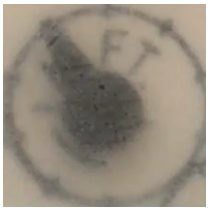
3. 3.06872

2. 2.12698

1. 1.15822

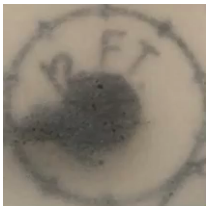


0. 0.0209366

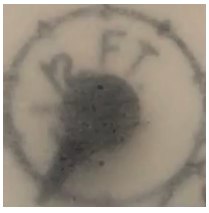


9. 8.99893

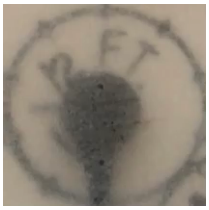
8. 7.94127



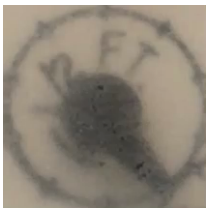
7. 6.91711



6. 5.93815

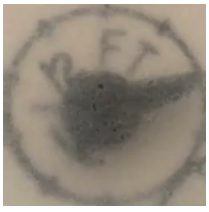


5. 5.09774



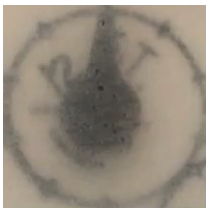
4. 4.05716

3. 3.19994



2. 2.16478

1. 1.18382



0. 9.93654

Looks very nice.

In[64]:=
```
{time, posns} = AbsoluteTiming[predict /@ frames];
time
Print[Length@frames / time, " fps."]
```

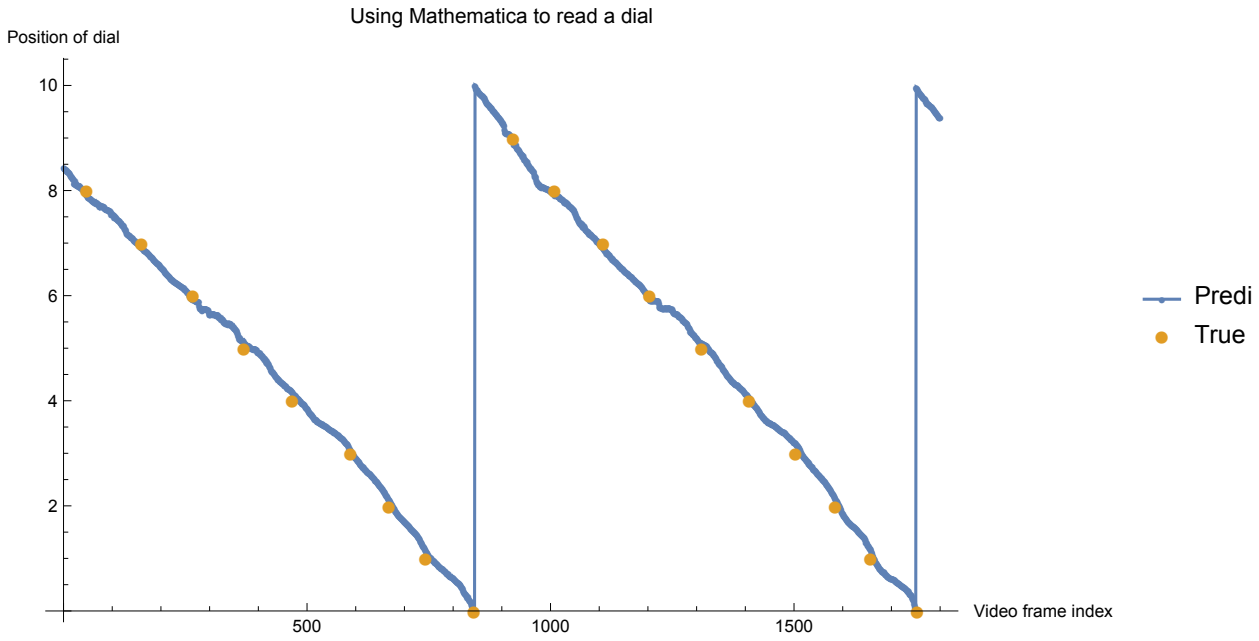
Out[65]= 13.801

During evaluation of In[64]:=

130.426 fps.

In[67]:=
```
plotClock1 = ListPlot[{posns, {traininginds, trainingclockpositions}^T},
  PlotMarkers → {{●, 5}, {●, 10}}, Joined -> { True, False},
  PlotLabel → "Using Mathematica to read a dial",
  PlotLegends → {"Predicted dial positions", "True position from training set"},
  AxesLabel → {"Video frame index", "Position of dial"}, ImageSize → Large]
```

Out[67]=



OLD: Predict dial *x*, *y* coordinates

I locked these cells because newer versions of MMA don't produce the same nice outputs we have here. I left this here as a cautionary tale -- if you use MMA's Predict[] function, and they change it later, it'll break your stuff.

To get around the 0->9 discontinuity, let's make two predictors: one for the *x* coordinate of the tip of the dial, and one for the *y* coordinate. These are continuous quantities of the angle, so perhaps this will be more amenable to prediction.

```
x = N@Sin[2 π clockpositions/10];
y = N@Cos[2 π clockpositions/10];
angles = 180/π ArcTan[x, y];
trainx = frames[[frameinds]] → x;
trainy = frames[[frameinds]] → y;

px = Predict[trainx, Method → "LinearRegression", PerformanceGoal → "Quality"];
py = Predict[trainy, Method → "LinearRegression", PerformanceGoal → "Quality"];
```

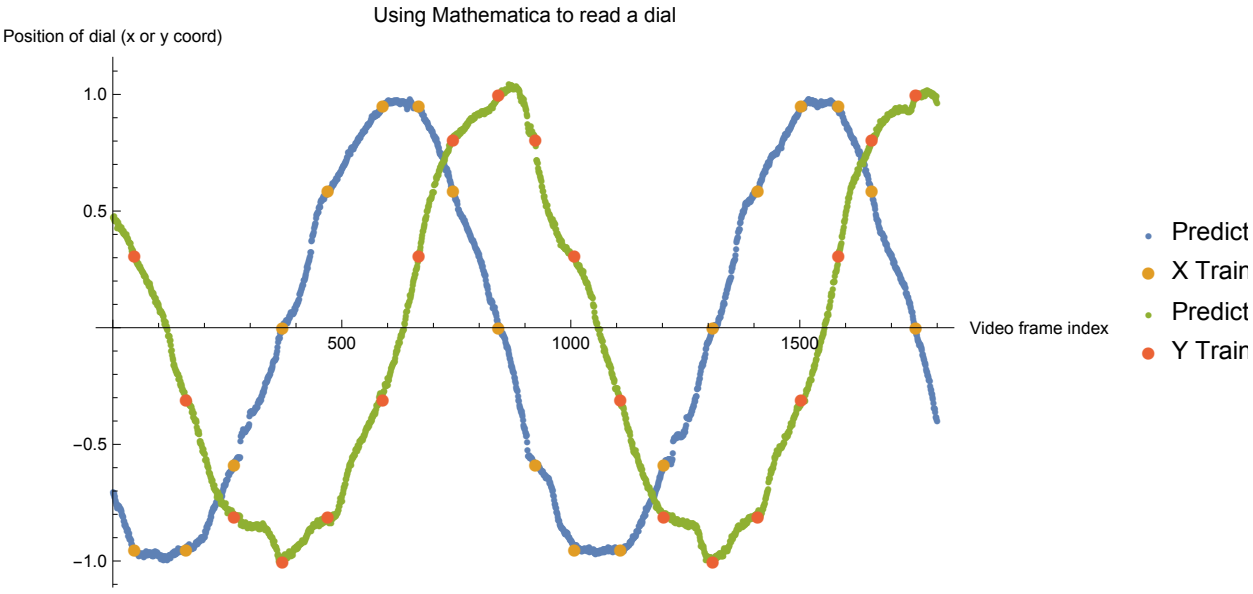
Let's see how well these do:

```
{time, xs} = AbsoluteTiming[px /@ frames];
time
{time, ys} = AbsoluteTiming[py /@ frames];
time
```

```
16.290651
```

```
16.302841
```

```
plot = ListPlot[{xs, {frameinds, x}ᵀ, ys, {frameinds, y}ᵀ},
  PlotMarkers → {{●, 5}, {●, 10}}, Joined -> { False, False},
  PlotLabel → "Using Mathematica to read a dial",
  PlotLegends → {"Predicted X", "X Training set", "Predicted Y", "Y Training set"},
  AxesLabel → {"Video frame index", "Position of dial (x or y coord)"},
  ImageSize → Large]
```




```
Export["dial_positions_xy.png", plot];
```

Armed with these, this function calculates the angle of the dial in degrees, then the dial position in [0,10):

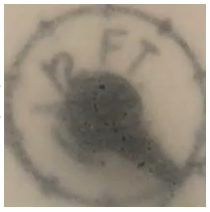
```
Clear[pAngleDeg, angle2clock]
```

$$\text{pAngleDeg} = \frac{180}{\pi}\,\text{ArcTan}[\text{px@\#}, \text{py@\#}]\ \&;$$

$$\text{angle2clock}[\theta\_] := \text{Mod}\left[\frac{10}{360}\,(90 - \theta), 10\right];$$

Do the predictor functions work?

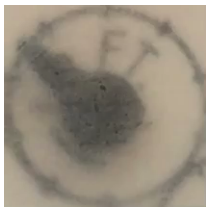
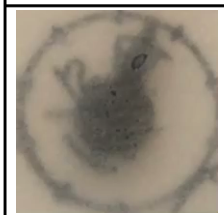
```
With[{f = frames[[500]]}, {f, pAngleDeg[f], angle2clock[pAngleDeg[f]]}]
```

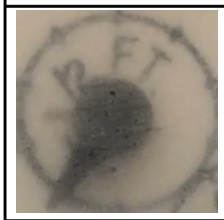
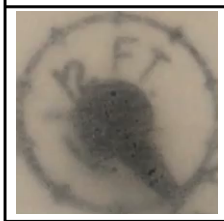


{ , -46.8076, 3.80021}

That looks like 4 o'clock degrees, nice.

```
tab =
  Table[{f, pAngleDeg[f], angle2clock[pAngleDeg[f]]}, {f, frames[[1 ;; -1 ;; 200]]}];
Grid[tab, Frame → All]
```

	145.817	8.44953
	-148.789	6.63304
	-83.7146	4.82541
	-14.2011	2.89448
	71.5818	0.511615

	160.953	8.02908
	−126.611	6.01697
	−53.9613	3.99892
	29.6138	1.6774

Predict the dial's angle for all zillion frames.

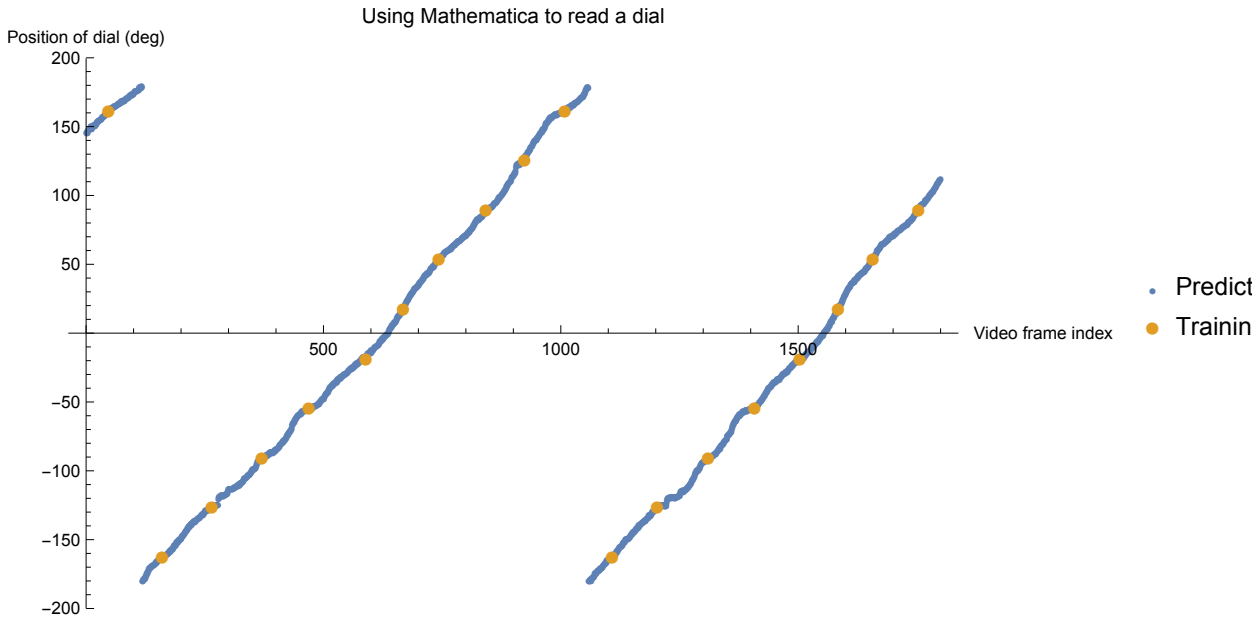
```
{time, allAngles} = AbsoluteTiming[pAngleDeg /@ frames];
time
```

$$\text{Print}\left[\frac{\text{Length@frames}}{\text{time}}, \text{" fps."}\right]$$

```
32.655972
```

```
55.12009 fps.
```

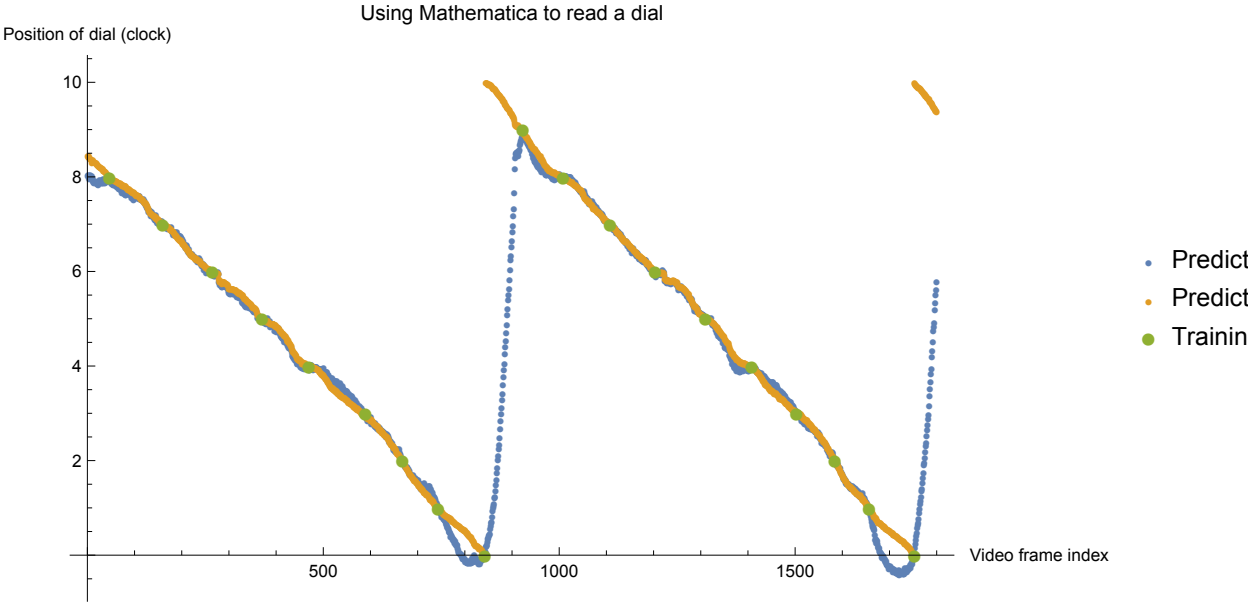
```
plot = ListPlot[{allAngles, {frameinds, angles}^T}, PlotMarkers → {{●, 5}, {●, 10}},
  Joined -> { False, False}, PlotLabel → "Using Mathematica to read a dial",
  PlotLegends → {"Predicted dial positions (deg)", "Training set"},
  AxesLabel → {"Video frame index", "Position of dial (deg)"}, ImageSize → Large]
```




```
Export["dial_positions_degrees.png", plot];
```

```
allClocks = angle2clock[allAngles];
```

```
plotClock2 = ListPlot[{posns, allClocks, {frameinds, clockpositions}^T},
  PlotMarkers → {{●, 5}, {●, 5}, {●, 10}}, Joined -> { False, False},
  PlotLabel → "Using Mathematica to read a dial",
  PlotLegends → {"Predicted dial positions (clock)",
    "Predicted dial positions w/ arctan (clock)", "Training set"},
  AxesLabel → {"Video frame index", "Position of dial (clock)"}, ImageSize → Large]
```



We've removed the discontinuity in the predictor! This is because the predictor is secretly predicting the x and y coords.

```
Export["dial_positions_clockhands2.png", plotClock2];
```