Thermocouple Calibration Report 2

Brian Carrozza
Last Updated: 2020-03-12 18:21:50

Intro

We used type T and type K thermocouples to collect temperature data. The following chart from https://www.thermocoupleinfo.com/thermocouple-accuracies.htm (https://www.thermocoupleinfo.com/thermocouple-accuracies.htm) shows the accuracy of these thermocouple types.

Thermocouple Conductor Type	Limits of Error (Whichever is greater)	
	Standard	Special
Type K	±2.2°C or ±0.75%	±1.1°C or ±0.4%
Type T	±1.0°C or ±0.75%	±0.5°C or ±0.4%

We will refer to these standards when evaluating our custom-made, opensource Data Aquisition (DAQ) system.

Description

Lucas and I built 2 "Thermo-Boards" which can collect 16 thermocouple temperatures each. Each board connects to one Raspberry Pi 3B+. The R-pi logs the data. We will refer to the assembly of a Thermo-Board and a R-pi as a DAQ. We only have 12 thermocouples connected to each DAQ, for a total of 24 temperature sensors. We conducted a calibration procedure where we submerged the TC's in boiling water for 2 minutes, then in ice water for 2 minutes. We cycled three times. The goal is to analyze the accuracy and precision of each TC and to develop a custom calculation to adjust for optimal accuracy. The accepted precision of thermocouples type T and K is ? and ? respectively.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# plt.rcParams['font.size'] = 14
# %matplotlib inline
```

In [2]:
```
data1 = pd.read_csv("outputlog_20200312/outputlog1_cold.csv")
data2 = pd.read_csv("outputlog_20200312/outputlog2_cold.csv")

data3 = pd.read_csv("outputlog_20200312/outputlog1_hot.csv")
data4 = pd.read_csv("outputlog_20200312/outputlog2_hot.csv")
```

In [3]:
```
# np.shape(data1)==np.shape(data2)
```

In [4]:
```
# np.shape(data3)==np.shape(data4)
```

Combine the data sets into one

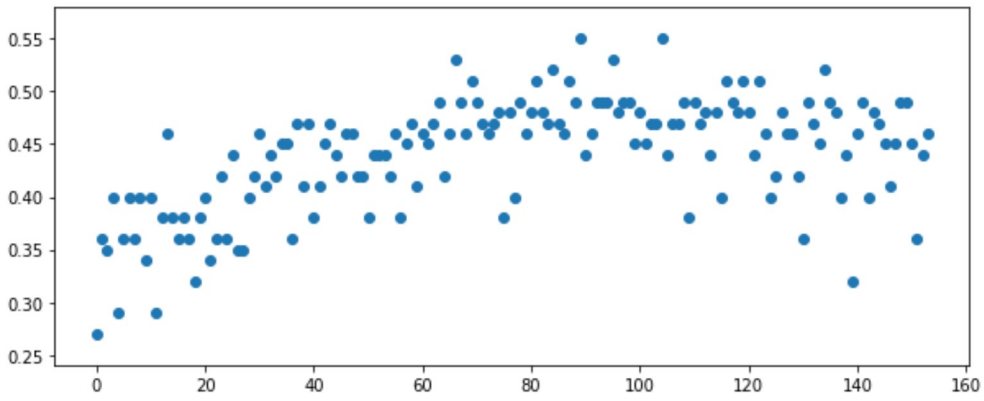
In [5]:
```
# In this case, a deep copy
data_cold = data1.copy()

# concatenate on axis=1 should also work if the date/times match
# but this is appending one column at a time
for key in data2:
    data_cold[key] = data2[key]


data_hot = data3.copy()

for key in data4:
    data_hot[key] = data4[key]

data_cold.iloc[:5, :16]
```

Out[5]:

	Date/Time 1	Tc 101	Tc 102	Tc 103	Tc 104	Tc 105	Tc 106	Tc 107	Tc 108	Tc 109	Tc 110	Tc 111	Tc 112	Tc 113	
0	12/03/2020 16:39:11	0.27	0.38	0.41	0.47	0.46	0.45	0.51	0.47	0.52	0.55	0.58	0.64	0.68	(
1	12/03/2020 16:39:25	0.36	0.41	0.41	0.45	0.49	0.52	0.44	0.58	0.49	0.51	0.63	0.59	0.73	(
2	12/03/2020 16:39:39	0.35	0.44	0.47	0.40	0.41	0.42	0.41	0.49	0.52	0.68	0.59	0.61	0.59	(
3	12/03/2020 16:39:54	0.40	0.40	0.42	0.49	0.41	0.47	0.49	0.40	0.53	0.59	0.64	0.61	0.58	(
4	12/03/2020 16:40:08	0.29	0.40	0.45	0.41	0.48	0.47	0.52	0.49	0.55	0.57	0.63	0.57	0.66	(

A quick scatterplot of one TC

In [6]:
```
def scat(y):
    """Scatter plot with 10 by 4 figure size."""

    x = range(len(y))
    plt.figure(figsize=(10, 4))
    plt.scatter(x, y)


scat(data_cold["Tc 101"])
```



Get rid of data from Cold Junction 1 and 2 and the Date/Time2

In [7]:
```
data_cold.drop(columns=["Cold Junction 1","Date/Time 2","Cold Junct
ion 2"],inplace=True)
data_hot.drop(columns=["Cold Junction 1","Date/Time 2","Cold Juncti
on 2"],inplace=True)
```

In [8]:
```
data_cold.iloc[:5,9:20]
```

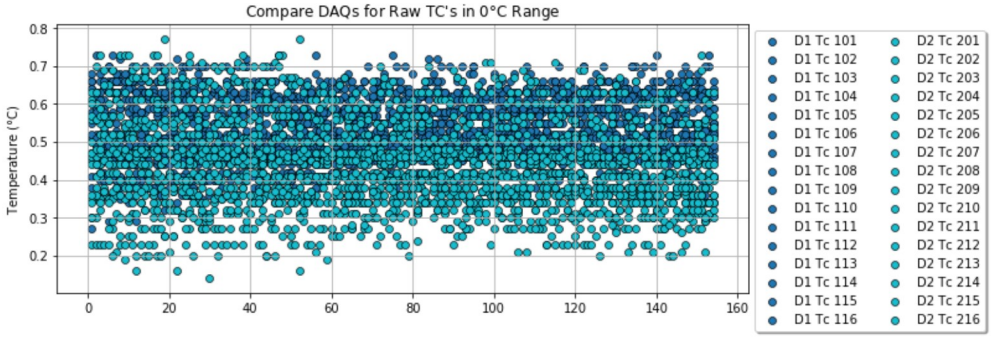
Out[8]:

	Tc 109	Tc 110	Tc 111	Tc 112	Tc 113	Tc 114	Tc 115	Tc 116	Tc 201	Tc 202	Tc 203
0	0.52	0.55	0.58	0.64	0.68	0.66	0.66	0.64	0.23	0.30	0.30
1	0.49	0.51	0.63	0.59	0.73	0.62	0.58	0.63	0.23	0.30	0.35
2	0.52	0.68	0.59	0.61	0.59	0.61	0.63	0.66	0.23	0.32	0.35
3	0.53	0.59	0.64	0.61	0.58	0.64	0.66	0.66	0.23	0.36	0.46
4	0.55	0.57	0.63	0.57	0.66	0.70	0.70	0.64	0.23	0.27	0.35

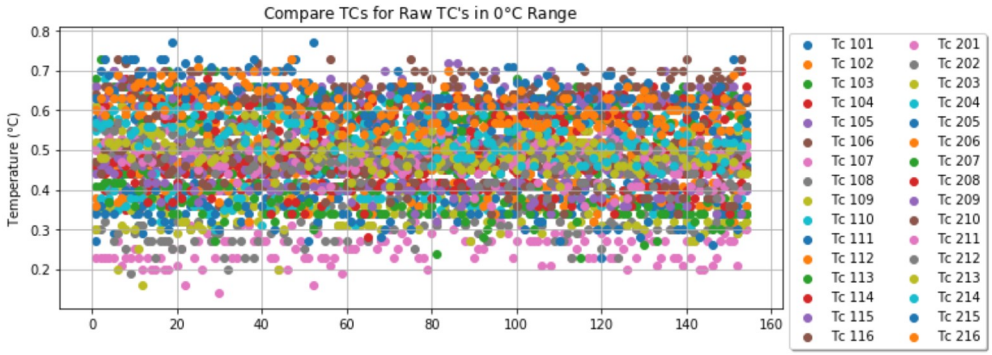
Raw TC's in the freezing range

Of both DAQs combined

In [9]:
```
data = data1
rg = range(1, len(data)+1)
x = list(rg)
fig, ax = plt.subplots(figsize=(10, 4))
for key in data.keys()[1:-1]:
    y = data[key]
    ax.scatter(x, y, label="D1 "+key, color='tab:blue', edgecolors=
'k',
               linewidths=.65)

data = data2
for key in data.keys()[1:-1]:
    y = data[key]
    ax.scatter(x, y, label="D2 "+key, color='tab:cyan',edgecolors='
k',
               linewidths=.65)

ax.legend(loc='upper left', ncol=2, fancybox=True, shadow=True,
          bbox_to_anchor=(1, 1))
ax.grid(True)
plt.title("Compare DAQs for Raw TC's in 0$\degree$C Range")
plt.ylabel("Temperature ($\degree$C) ")
plt.show()
```



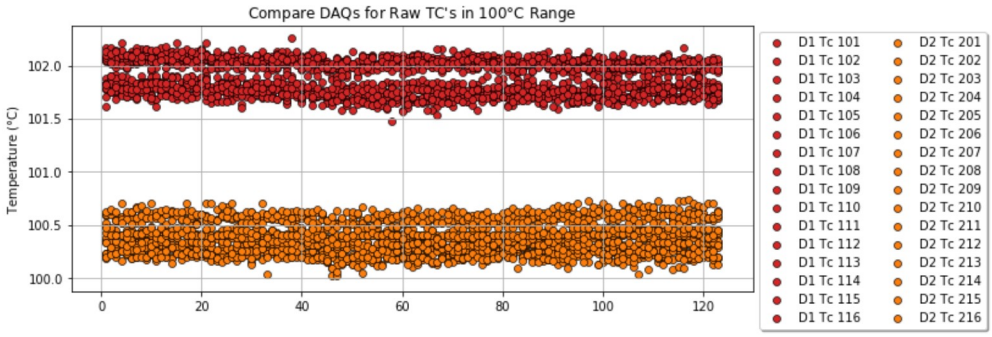
In [10]:
```
data = data_cold
rg = range(1, len(data)+1)
x = list(rg)
fig, ax = plt.subplots(figsize=(10, 4))
for key in data.keys()[1:]:
    y = data[key]
    ax.scatter(x, y, label=key)

ax.legend(loc='upper left', ncol=2, fancybox=True, shadow=True,
          bbox_to_anchor=(1, 1))
ax.grid(True)
plt.title("Compare TCs for Raw TC's in 0$\degree$C Range")
plt.ylabel("Temperature ($\degree$C) ")
plt.show()
```



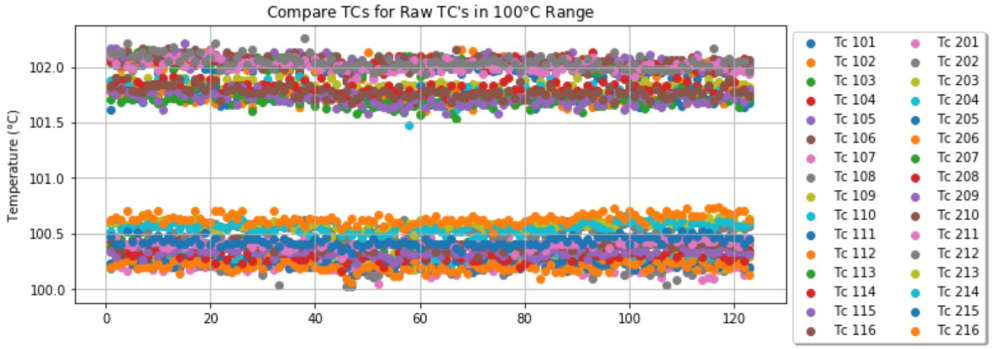
Raw TC's in the boiling range

Here we can see that the 100^{o}C TC's have a gap between the two DAQs.

In [11]:
```
data = data3
rg = range(1, len(data)+1)
x = list(rg)
fig, ax = plt.subplots(figsize=(10, 4))
for key in data.keys()[1:-1]:
    y = data[key]
    ax.scatter(x, y, label="D1 "+key, color='tab:red', edgecolors='
k',
               linewidths=.65)

data = data4
for key in data.keys()[1:-1]:
    y = data[key]
    ax.scatter(x, y, label="D2 "+key, color='tab:orange', edgecolor
s='k',
               linewidths=.65)

ax.legend(loc='upper left', ncol=2, fancybox=True, shadow=True,
          bbox_to_anchor=(1, 1))
ax.grid(True)
plt.title("Compare DAQs for Raw TC's in 100$\degree$C Range")
plt.ylabel("Temperature ($\degree$C) ")
plt.show()
```



In [12]:
```
data = data_hot
rg = range(1, len(data)+1)
x = list(rg)
fig, ax = plt.subplots(figsize=(10, 4))
for key in data.keys()[1:]:
    y = data[key]
    ax.scatter(x, y, label=key)

ax.legend(loc='upper left', ncol=2, fancybox=True, shadow=True,
          bbox_to_anchor=(1, 1))
ax.grid(True)
plt.title("Compare TCs for Raw TC's in 100$\degree$C Range")
plt.ylabel("Temperature ($\degree$C) ")
plt.show()
```



Finding the np.mean and stdev of the cold data

In [13]:
```
stats = pd.DataFrame({"TC": data.keys()[1:]})
stats["T_cold_avg"] = 0
stats["T_hot_avg"] = 0
stats["T_cold_stdev_smpl"] = 0
stats["T_hot_stdev_smpl"] = 0

stats.drop(columns=["TC"], inplace=True)
stats.index = data.keys()[1:]

stats.loc[:, "T_hot_avg"] = np.mean(data_hot)
stats.loc[:, "T_cold_avg"] = np.mean(data_cold)
stats.loc[:, "T_hot_stdev_smpl"] = np.std(data_hot, ddof=1)
stats.loc[:, "T_cold_stdev_smpl"] = np.std(data_cold, ddof=1)

x = stats.T_cold_stdev_smpl.max()
y = stats.T_hot_stdev_smpl.max()

print("""The maximum deviation for cold and hot,
shown as ±3σ, are ±{0} and ±{1}, respectively.""".format(
    round(x*3, 2), round(y*3, 2)))
print("""\nThese values belong to thermocouples {0} and {1}.""".for
mat(
    stats[stats.T_cold_stdev_smpl == x].index[0],
    stats[stats.T_hot_stdev_smpl == y].index[0]))
```

```
The maximum deviation for cold and hot,
shown as ±3σ, are ±0.16 and ±0.19, respectively.

These values belong to thermocouples Tc 101 and Tc 110.
```

All thermocouples on DAQ1 are type K.
For DAQ2, thermocouples 201 through 204 are type K and the rest are type T.

These deviations are within the accuracy range of "Special" thermocouples for their repspective types.

Thermocouple Conductor Type	Limits of Error (Whichever is greater)	
	Standard	Special
Type K	±2.2°C or ±0.75%	±1.1°C or ±0.4%
Type T	±1.0°C or ±0.75%	±0.5°C or ±0.4%

This is a sample of the table with the mean temperatures at boiling and freezing and the standard deviation at hot and cold.

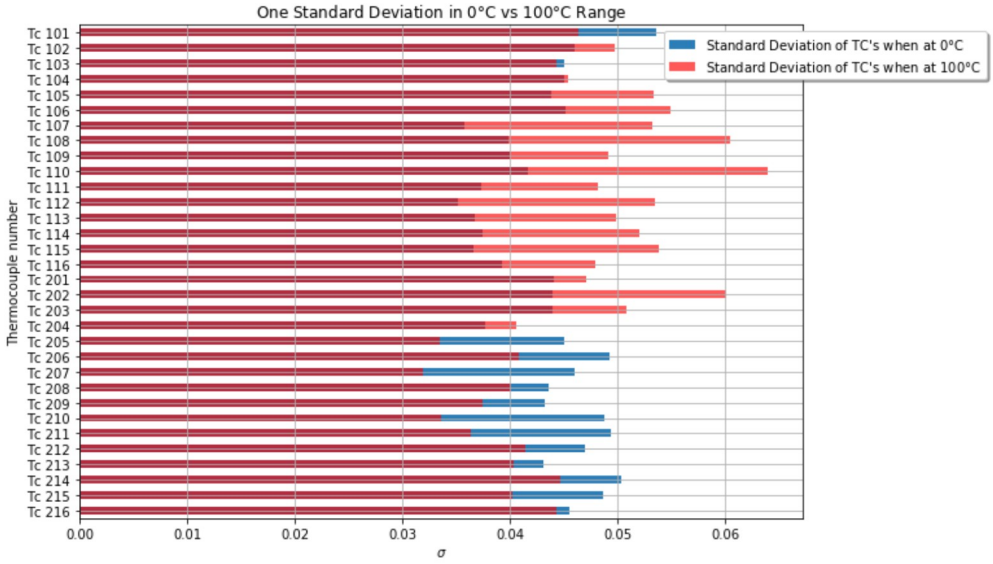
In [14]:
```
stats.head()
```

Out[14]:

	T_cold_avg	T_hot_avg	T_cold_stdev_smpl	T_hot_stdev_smpl
Tc 101	0.440130	101.996667	0.053639	0.046424
Tc 102	0.464610	102.036504	0.046020	0.049786
Tc 103	0.471818	102.036016	0.045034	0.044311
Tc 104	0.469545	102.037398	0.045047	0.045446
Tc 105	0.481039	102.050000	0.043823	0.053408

Barchart of TC standard deviation

In [15]:
```
# stats[stats.T_hot_stdev_smpl.max() == stats.T_hot_stdev_smpl]


fig, ax = plt.subplots(figsize=(10, 7))
stats.T_cold_stdev_smpl.plot(
    kind="barh", color='tab:blue', alpha=0.95,
    label="Standard Deviation of TC's when at 0$\degree$C")
stats.T_hot_stdev_smpl.plot(
    kind="barh", alpha=0.65, color='r',
    label="Standard Deviation of TC's when at 100$\degree$C")
plt.title("One Standard Deviation in 0$\degree$C vs 100$\degree$C R
ange")
plt.xlabel("$\sigma$")
plt.ylabel("Thermocouple number")
plt.gca().invert_yaxis()
ax.legend(loc='upper left', ncol=1, fancybox=True, shadow=True,
          bbox_to_anchor=(.8, 1))
plt.grid(True)
```



Determine the linear adjustment for each TC

Using $y = mx + b$, adjust the accuracy of each TC
The table below include the values of m and b

In [16]:
```
stats["m"] = 0; stats["b"] = 0

for rkey in stats.index:
    c_avg = stats.loc[rkey, "T_cold_avg"]
    h_avg = stats.loc[rkey, "T_hot_avg"]
    stats.loc[rkey, "m"], stats.loc[rkey, "b"] = np.linalg.solve(
        [[h_avg, 1], [c_avg, 1]], [[100], [0]])

stats.head(9)
```

Out[16]:

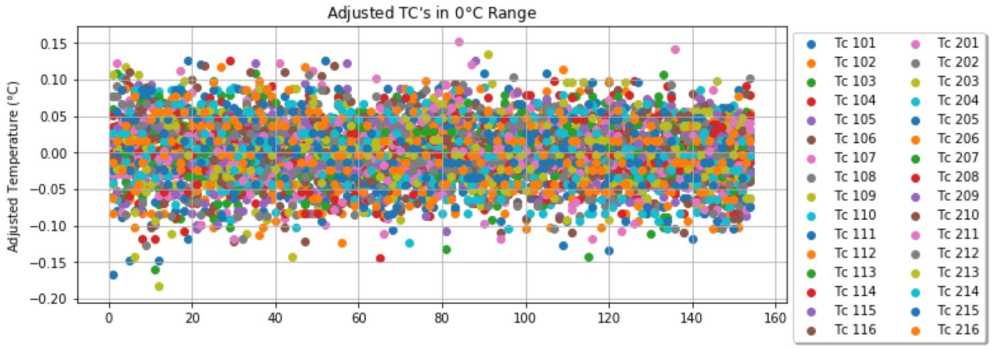
	T_cold_avg	T_hot_avg	T_cold_stdev_smpl	T_hot_stdev_smpl	m	b
Tc 101	0.440130	101.996667	0.053639	0.046424	0.984673	-0.433384
Tc 102	0.464610	102.036504	0.046020	0.049786	0.984524	-0.457420
Tc 103	0.471818	102.036016	0.045034	0.044311	0.984599	-0.464552
Tc 104	0.469545	102.037398	0.045047	0.045446	0.984563	-0.462297
Tc 105	0.481039	102.050000	0.043823	0.053408	0.984553	-0.473608
Tc 106	0.483312	102.047724	0.045202	0.054949	0.984597	-0.475867
Tc 107	0.492013	102.013333	0.035817	0.053265	0.985015	-0.484640
Tc 108	0.487078	102.047642	0.039835	0.060423	0.984634	-0.479594
Tc 109	0.493377	101.827398	0.039987	0.049187	0.986835	-0.486882

Apply the formulas to each TC

In [17]:
```
data_cold2 = data_cold.copy()
for rkey in stats.index:
    data_cold2[rkey] = data_cold2[rkey] * \
        stats.loc[rkey, "m"]+stats.loc[rkey, "b"]
```

Graph the accuracy-adjusted values to see efficacy on TC's at 0 deg C

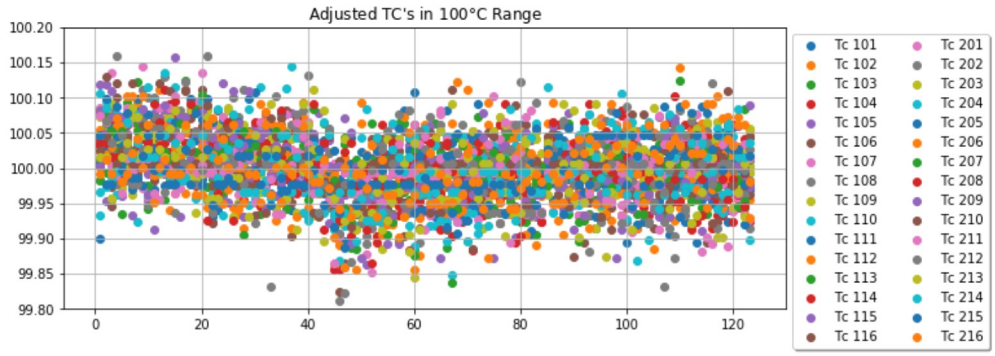
In [18]:
```
data = data_cold2
rg = range(1, len(data)+1)
x = list(rg)
fig, ax = plt.subplots(figsize=(10, 4))
for key in data.keys()[1:]:
    y = data[key]
    ax.scatter(x, y, label=key)

ax.legend(loc='upper left', ncol=2, fancybox=True, shadow=True,
          bbox_to_anchor=(1, 1))
ax.grid(True)
plt.title("Adjusted TC's in 0$\degree$C Range")
plt.ylabel("Adjusted Temperature ($\degree$C) ")
plt.show()
```



It looks like it worked very well on the training set.

Graph the accuracy-adjusted values to see efficacy on TC's at 100 deg C

In [19]:
```
data_hot2 = data_hot.copy()
for rkey in stats.index:
    data_hot2[rkey] = data_hot2[rkey]*stats.loc[rkey, "m"]+stats.lo
c[rkey, "b"]
data = data_hot2
rg = range(1, len(data)+1)
x = list(rg)
fig, ax = plt.subplots(figsize=(10, 4))
for key in data.keys()[1:]:
    y = data[key]
    ax.scatter(x, y, label=key)

ax.legend(loc='upper left', ncol=2, fancybox=True, shadow=True,
          bbox_to_anchor=(1, 1))
ax.grid(True)
plt.title("Adjusted TC's in 100$\degree$C Range")
plt.ylim(99.8,100.2)
plt.show()
```



Both training sets are much more accurate than before.
How much more?

Take the $\frac{[raw\ mean\ cold]-0}{100}$

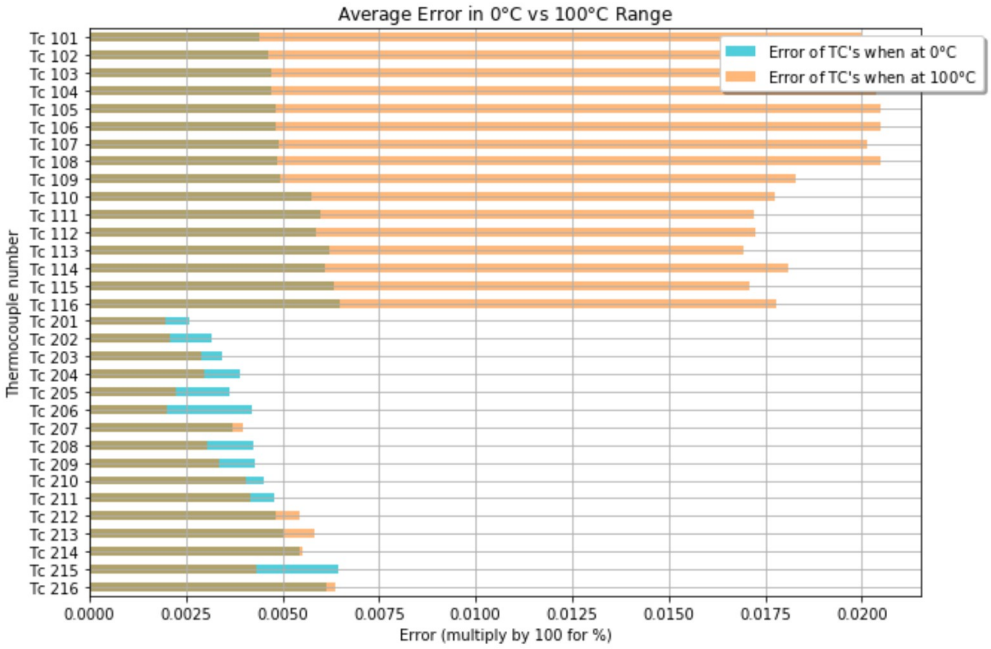
and the $\frac{[raw\ mean\ hot]-100}{100}$

In [20]:
```
stats["Error_cold"]=abs(stats.T_cold_avg/100)
stats["Error_hot"]=abs((stats.T_hot_avg-100)/100)
stats.head()
```

Out[20]:

	T_cold_avg	T_hot_avg	T_cold_stdev_smpl	T_hot_stdev_smpl	m	b
Tc 101	0.440130	101.996667	0.053639	0.046424	0.984673	-0.433384
Tc 102	0.464610	102.036504	0.046020	0.049786	0.984524	-0.457420
Tc 103	0.471818	102.036016	0.045034	0.044311	0.984599	-0.464552
Tc 104	0.469545	102.037398	0.045047	0.045446	0.984563	-0.462297
Tc 105	0.481039	102.050000	0.043823	0.053408	0.984553	-0.473608

In [21]:
```
fig, ax = plt.subplots(figsize=(10, 7))
stats.Error_cold.plot(
    kind="barh", color='tab:cyan', alpha=0.75,
    label="Error of TC's when at 0$\degree$C")
stats.Error_hot.plot(
    kind="barh", alpha=0.55, color='tab:orange',
    label="Error of TC's when at 100$\degree$C")
plt.title("Average Error in 0$\degree$C vs 100$\degree$C Range")
plt.xlabel("Error (multiply by 100 for %)")
plt.ylabel("Thermocouple number")
ax.legend(loc='upper left', ncol=1, fancybox=True, shadow=True,
          bbox_to_anchor=(.75, 1))
plt.gca().invert_yaxis()
plt.grid(True)
```



Here we can see that DAQ1 is varying more than DAQ2. Why? This is difficult to explain because they are constructed the same and located in the same place. The good new is that both DAQs are within spec for ±0.5°C.

Conclusion

It looks like our DAQs are reliable enough to collect accurate and precise data.