Handled VESAD Analysis

March 13, 2018

The analysis of the experiment data is also accessible online: https://github.com/NormandErwan/master-thesis-analysis/blob/master/Handled%20VESAD%20Analysis.ipynb.

1 1. Data preparation

Configuration :

```
In [1]: import numpy as np
        import pandas as pd
        from pandas.api.types import CategoricalDtype
        import itertools

        import seaborn as sns
        import matplotlib.pyplot as plt
        import matplotlib.ticker as ticker

        from scipy import stats
        import statsmodels.api as sm
        from statsmodels.formula.api import ols
        from statsmodels.stats.multicomp import (MultiComparison, pairwise_tukeyhsd)
        from statsmodels.stats.multitest import multipletests
        from statsmodels.stats.libqsturng import psturng

        from ast import literal_eval
        from os import listdir
        from os.path import join
```

```
C:\Users\Erwan\Miniconda3\envs\master-thesis\lib\site-packages\statsmodels\compat\pandas.py:56: FutureWa
  from pandas.core import datetools
```

```
In [2]: %matplotlib inline

        sns.set_style('whitegrid')
        sns.set_context('notebook')

        # Render the plots with this language
        language = 'English'
        #language = 'Français'
```

Data are loaded in the following variabbles: - Participants information, from the questionary before the trials: `participants` - The summary of the trials of the participants: `trials` - The detailed measures of the trials: `trial_details` - The ranks of the participants from the post-questionary: `ranks`

```
In [3]: participants = pd.read_csv('participants.csv')
        trials = pd.read_csv('participant_trials.csv')
        trials_for_anova = pd.read_csv('participant_trials.csv')
        ranks = pd.read_csv('participant_ranks.csv')
```

```
In [4]: # Detailled trials are long to load, keep commented if unused

        #trial_details = []
        #for file in listdir('.'):
        #    if file.endswith('details.csv'): # The trial files end with *-details.csv
        #        details.append(pd.read_csv(file))
        #trial_details = pd.concat(details, ignore_index=True)
```

Creates independent variables lists (`participants_ivs`, `trials_ivs`) and dependent variables lists (`ranks_dvs`, `trials_dvs`) to make easier to use algorithms and to plot figures.

```
In [5]: # Participants IVs
        if language == 'Français':
            participants_iv_labels = ['Numéro participant', 'Sexe', 'Porte des lunettes',\
                                      'Porte des lentilles', 'Est daltonien',\
                                      'Intervalle d\'âge', 'Main dominante',\
                                      'Main utilisée pour la souris', 'Activité principale',\
                                      'Utilisation de l\'ordinateur (heures/jours)',\
                                      'Logiciels 3D utilisés', 'Visiocasques RV/RA utilisés',\
                                      'Techniques d\'interactions RV/RA utilisées']
        else:
            participants_iv_labels = ['Participant Id', 'Sex', 'Has Glasses',\
                                      'Has Contact Lenses', 'Is Color Blind',\
                                      'Age Class', 'Dominant Hand',\
                                      'Hand Used for Mouse', 'Activity',\
                                      'Computer Hours per Day', '3D Softwares Used',\
                                      'HMD Used', 'Known Interactions Techniques on HMD']

        participants_ivs = pd.Series(data=participants_iv_labels, index=participants.columns)
```

```
In [6]: # Trials IVs
        if language == 'Français':
            trials_iv_labels = ['Technique', 'Taille du texte', 'Distance', 'Groupe',\
                                'Méthode d\'entrée', 'Méthode d\'affichage']
        else:
            trials_iv_labels = ['Technique', 'Text Size', 'Distance', 'Ordering',\
                                'Technique Input', 'Technique Output']

        trials_ivs = ['technique', 'text_size', 'distance', 'ordering', 'input', 'output']
        trials_ivs = pd.DataFrame(columns=trials_ivs, index=['label', 'categorical', 'palette'])

        default_palette = sns.color_palette('Set2', 8)

        for iv_index, iv_label in zip(trials_ivs.columns, trials_iv_labels):
```

```
            iv_categories = trials.sort_values([iv_index + '_id']).drop_duplicates(iv_index)[iv_index]
            iv_categorical = pd.Categorical(iv_categories, iv_categories, ordered=True)
            trials_ivs[iv_index] = [iv_label, iv_categorical, default_palette]

        trials_ivs.at['palette', 'technique'] = [default_palette[1], default_palette[0], default_palett
        trials_ivs.at['palette', 'text_size'] = sns.light_palette(default_palette[6], 3)[1:3] # Brown p
        trials_ivs.at['palette', 'distance'] = sns.light_palette(default_palette[4], 3)[1:3] # Green pa
        trials_ivs.at['palette', 'ordering'] = sns.light_palette(default_palette[3], 4)[1:4] # Pink tri
        trials_ivs.at['palette', 'input'] = [default_palette[5], default_palette[2]]
        trials_ivs.at['palette', 'output'] = [default_palette[0], sns.color_palette('muted')[5]]

        if language == 'Français':
            trials_ivs.at['categorical', 'technique'].categories = ['Téléphone seul',\
                                                                    'Téléphone étendu tactile',\
                                                                    'Téléphone étendu touché autour']
            trials_ivs.at['categorical', 'text_size'].categories = ['Grand', 'Petit']
            trials_ivs.at['categorical', 'distance'].categories = ['Proche', 'Loin']
            trials_ivs.at['categorical', 'ordering'].categories = ['Groupe 1', 'Groupe 2', 'Groupe 3']
            trials_ivs.at['categorical', 'input'].categories = ['Tactile', 'Autour du téléphone']
            trials_ivs.at['categorical', 'output'].categories = ['Téléphone seul', 'Téléphone étendu']
```

```
In [7]: technique, text_size, distance = trials_ivs['technique'], trials_ivs['text_size'], trials_ivs['
        ordering, iv_input, iv_output =  trials_ivs['ordering'], trials_ivs['input'], trials_ivs['outpu
```

```
In [8]: # Shortcut to access the different trials IVs' categories
        categories = pd.Series([trials_ivs[iv_index]['categorical'].categories for iv_index in trials_i
```

```
In [9]: # Trials DVs
        if language == 'Français':
            trials_dv_labels = ['Temps de complétion (s)', 'Sélections', 'Temps de sélection',\
                                 'Distance 3D de sélection', 'Distance de sélection',\
                                 'Déselections', 'Erreurs', 'Disques classés', 'Défilements',\
                                 'Temps de défilement', 'Distance 3D de défilement',\
                                 'Distance de défilement', 'Zooms', 'Temps de zoom', 'Distance 3D de zoom
                                 'Distance de zoom', 'Distance tête-téléphone',\
                                 'Distance relative tête-téléphone']
        else:
            trials_dv_labels = ['Task Completion Time (s)', 'Selections', 'Selection Time',\
                                'Selection Distance', 'Selection Distance on Grid',\
                                'Deselections', 'Errors', 'Items Classified', 'Pans', 'Pan Time',\
                                'Pan Distance', 'Pan Distance on Grid',\
                                'Zooms', 'Zoom Time', 'Zoom Distance',\
                                'Zoom Distance on Grid', 'Head Phone Distance',\
                                'Signed Head Phone Distance']

        trials_dvs = trials.loc[:, 'total_time':'signed_head_phone_distance'].columns
        trials_dvs = pd.Series(data=trials_dv_labels, index=trials_dvs)
```

```
In [10]: # Ranks DVs
         if language == 'Français':
             ranks_dv_labels = ['Facile à comprendre', 'Mentalement facile à utiliser',\
                                'Physiquement facile à utiliser', 'Rapidité', 'Performance',\
                                'Frustration', 'Préférence']
         else:
```

```
    ranks_dv_labels = ['Easy to Understand', 'Mentally Easy to Use',\
                       'Physically Easy to Use', 'Subjective Speed',\
                       'Subjective Performance', 'Frustration', 'Preference']

ranks_dv_scales = [pd.Categorical(list(range(1,6)), list(range(1,6)), ordered=True)]  * len(ra
ranks_dv_palettes = [sns.color_palette('RdYlBu', 5)] * len(ranks_dv_labels)

ranks_dvs = ranks.loc[:, 'easy_understand':'preference'].columns
ranks_dvs = pd.DataFrame(data=[ranks_dv_labels, ranks_dv_scales, ranks_dv_palettes],\
                         columns=ranks_dvs, index=['label', 'scale', 'palette'])

ranks_dvs.at['scale', 'preference'] = pd.Categorical(list(range(1,4)), list(range(1,4)), order
ranks_dvs.at['palette', 'preference'] = [sns.color_palette('RdYlBu', 5)[i] for i in range(4,-1
```

Clean the data:

```
In [11]: # Set better and translated columns to participants, trials and ranks
         participants.columns = participants_ivs

         columns = []
         for column in trials.columns:
             if (column in participants_ivs.index):
                 columns.append(participants_ivs[column])
             elif (column in trials_ivs.columns):
                 columns.append(trials_ivs[column]['label'])
             elif (column in trials_dvs.index):
                 columns.append(trials_dvs[column])
             else:
                 columns.append(column)

         trials.columns = columns

         ranks.columns = [participants_ivs['participant_id'], trials_ivs['ordering']['label'], trials_i
                         + ranks_dvs.loc['label', :].tolist()
```

```
In [12]: # Set the participant_id column as the index in participants
         participants.set_index(participants_ivs['participant_id'], inplace=True)
```

```
In [13]: # Some participants are non valid or don't have complete measures
         non_valid_participants = [0]
         participants = participants[~participants.index.isin(non_valid_participants)]
         ranks = ranks[~ranks[participants_ivs['participant_id']].isin(non_valid_participants)]

         incomplete_trials_participant_ids = [0, 4]
         trials = trials[~trials[participants_ivs['participant_id']].isin(incomplete_trials_participant
         trials_for_anova = trials_for_anova[~trials_for_anova['participant_id'].isin(incomplete_trials
```

```
In [14]: # Some participants have wrong head phone mesures
         for head_distance_column in ['absolute_head_phone_distance', 'signed_head_phone_distance']:
             trials.loc[trials[trials_dvs[head_distance_column]] == 0, trials_dvs['absolute_head_phone_
             trials_for_anova.loc[trials_for_anova[head_distance_column] == 0, head_distance_column] = 
```

```
In [15]: # Setup categorical columns participants, trials and ranks
         participants[trials_ivs['ordering']['label']] = ranks.groupby(participants_ivs['participant_id
```

```
participants[trials_ivs['ordering']['label']] = participants[trials_ivs['ordering']['label']].a

for iv_index in trials_ivs.columns:
    iv = trials_ivs[iv_index]
    trials_for_anova[iv_index] = trials[iv['label']] = trials[iv['label']].astype(iv['categori
    trials_for_anova[iv_index].cat.categories = trials[iv['label']].cat.categories = iv['catego

for iv_index in ['ordering', 'technique']:
    ranks[trials_ivs[iv_index]['label']] = ranks[trials_ivs[iv_index]['label']].astype(trials_i
```

In [16]:
```
# Eval the arrays in some dvs
def eval_if_str(data):
    return literal_eval(data) if isinstance(data, str) else data

trials['grid_config'] = trials['grid_config'].apply(eval_if_str)
```

Utilities:

In [17]:
```
def mask(df, f):
    return df[f(df)]

pd.DataFrame.mask = mask
```

In [18]:
```
def p_values_correction(data, p_value_label='p-value', alpha=0.05, correction_method='fdr_bh')
    if correction_method != None:
        reject, p_values_corrected, a1, a2 = multipletests(data[p_value_label].tolist(), alpha=
                                                           method=correction_method)
        data[p_value_label] = p_values_corrected
```

In [19]:
```
def subplots(nsubplots, ncols_max=3, subplotsize=(6,5), *plt_args):
    ncols = min(ncols_max, nsubplots)
    nrows = ((nsubplots - 1) // ncols) + 1
    fig, axs = plt.subplots(nrows=nrows, ncols=ncols, figsize=(subplotsize[0] * ncols, subplots
                            *plt_args)

    if nrows == 1 and ncols == 1:
        axs = [axs]
    elif nrows >= 2 and ncols >= 2:
        axs = [ax for ax_row in axs for ax in ax_row]

    for ax in axs[::-1][0:len(axs) - nsubplots]:
        fig.delaxes(ax)

    return (fig, axs)
```

In [20]:
```
labels = pd.Series()

if language == 'Français':
    labels['category'] = 'Catégorie'
    labels['count'] = 'Nombre'
    labels['distance'] = 'Distance moyenne (s)'
    labels['mean_rank'] = 'Note moyenne'
    labels['participants'] = 'Nombre de participants'
    labels['question'] = 'Question'
```

```
        labels['rank'] = 'Note'
        labels['time'] = 'Temps moyen (s)'
    else:
        labels['category'] = 'Category'
        labels['count'] = 'Count'
        labels['distance'] = 'Mean Distance (m)'
        labels['mean_rank'] = 'Mean Rank'
        labels['participants'] = 'Number of Participants'
        labels['question'] = 'Question'
        labels['rank'] = 'Rank'
        labels['time'] = 'Mean Time (s)'
```

2 2. Participant ranks

Some functions for the analysis:

```
In [21]: def get_ranks_count(iv_index, dv_index):
             iv, dv = trials_ivs[iv_index], ranks_dvs[dv_index]

             ranks_counts_index = pd.MultiIndex.from_product([iv['categorical'], dv['scale']],\
                                                             names=[iv['label'], labels['rank']])
             zero_ranks_counts = pd.Series(0, index=ranks_counts_index) # Zero counts by default
             ranks_counts = ranks.groupby([iv['label'], dv['label']]).size() # Gets the counts

             ranks_counts = pd.concat([ranks_counts, zero_ranks_counts]) # Merge counts with the defaul
             ranks_counts = ranks_counts[~ranks_counts.index.duplicated(keep='first')] # Keeps the coun
             ranks_counts.sort_index(inplace=True)
             ranks_counts.index = ranks_counts_index # Restore the index
             return ranks_counts

In [22]: def cumulated_barplot(data, palette, **args):
             for row_index, row in data.iloc[::-1].iterrows():
                 sns.barplot(x=data.columns, y=row, label=row_index, color=palette[row_index-1], **args)

In [23]: def plot_ranks_distributions(iv_index, dv_indexes):
             iv = trials_ivs[iv_index]

             fig, axs = subplots(len(dv_indexes))
             for dv_index, ax in zip(dv_indexes, axs):

                 dv = ranks_dvs[dv_index]

                 cumulated_ranks_count = get_ranks_count(iv_index, dv_index).unstack(level=0).cumsum()
                 cumulated_barplot(cumulated_ranks_count, palette=dv['palette'], ax=ax)
                 ax.set(ylabel=labels['participants'])

                 ax_handles, ax_labels = ax.get_legend_handles_labels()
                 ax.legend(ax_handles[::-1], ax_labels[::-1], frameon=True, loc='lower center', bbox_to
                           ncol=len(dv['scale']), title=dv['label'])

             fig.tight_layout(h_pad=4) # Add padding to avoir legend and labels overlap
             return (fig, axs)
```

In [24]:
```
def plot_ranks(iv_index, dv_indexes):
    iv = trials_ivs[iv_index]

    fig, axs = subplots(len(dv_indexes))
    for dv_index, ax in zip(dv_indexes, axs):

        dv = ranks_dvs[dv_index]

        sns.barplot(x=iv['label'], y=dv['label'], palette=iv['palette'], data=ranks, ax=ax)
        ax.set(ylim=(0, dv['scale'][-1]))
        ax.yaxis.set_major_locator(ticker.MultipleLocator(1)) # Fix the axis ticks

    return (fig, axs)
```

In [25]:
```
def test_ranks(iv_index, dv_indexes, **args):
    iv = trials_ivs[iv_index]

    results = []
    for dv_index in dv_indexes:
        dv = ranks_dvs[dv_index]
        samples = [ranks[ranks[iv['label']] == iv_value][dv['label']] for iv_value in iv['categ

        kruskal_H, p_value = stats.kruskal(*samples) # Compare the samples
        results.append([iv['label'], dv['label'], kruskal_H, p_value])

    results = pd.DataFrame(results, columns=['Independent Variable', 'Dependent Variable',\
                                              'Kruskal-Wallis H', 'p-value'])
    p_values_correction(results, **args)
    return results
```

In [26]:
```
def test_pairwise_ranks(iv_index, dv_indexes, **args):
    iv = trials_ivs[iv_index]
    iv_category_ids = range(len(iv['categorical']))

    results = []
    for dv_index in dv_indexes:
        dv = ranks_dvs[dv_index]
        samples = [ranks[ranks[iv['label']] == iv_value][dv['label']] for iv_value in iv['categ

        for iv_value_ids in itertools.combinations(iv_category_ids, 2): # Test each pair
            U, p_value = stats.mannwhitneyu(samples[iv_value_ids[0]], samples[iv_value_ids[1]]
            results.append([iv['label'], iv['categorical'][iv_value_ids[0]], iv['categorical']
                            dv['label'], U, p_value])

    results = pd.DataFrame(results, columns=['Independent Variable', 'Independent Variable Val
                                              'Independent Variable Value 2', 'Dependent Variab
                                              'Mann-Whitney U', 'p-value'])
    p_values_correction(results, **args)
    return results
```

Display the note distributions for each question and each technique.

In [27]:
```
(fig, axs) = plot_ranks_distributions('technique', ranks_dvs.columns)
```

We use Kruskal-Wallis test (Benjamini–Hochberg correction) on each question to check if there is significative differences between the ranks among TECHNIQUE:

```
In [28]: test_ranks('technique', ranks_dvs, correction_method='fdr_bh')
```

Out[28]:

	Independent Variable	Dependent Variable	Kruskal-Wallis H	p-value
0	Technique	Easy to Understand	10.811408	0.007859
1	Technique	Mentally Easy to Use	12.638068	0.004204
2	Technique	Physically Easy to Use	5.772681	0.055780
3	Technique	Subjective Speed	8.047606	0.025039
4	Technique	Subjective Performance	15.505873	0.001874
5	Technique	Frustration	6.304823	0.049874
6	Technique	Preference	15.065089	0.001874

All the questions, except Physically Easy to Use, have significant ranks among TECHNIQUES : Easy to Understand (p=0.0078), Mentally Easy to Use (p=0.0042), Subjective Speed (p=0.025), Subjective Performance (p=0.0018), Frustration (p=0.050), Preference (p=0.0018).

We use then pairwise Mann-Whitney test (Benjamini–Hochberg correction) for the significant question above:

```
In [29]: test_pairwise_ranks('technique', ['easy_understand', 'mentally_easy_use', 'could_go_fast',\
                                          'subjective_performance', 'preference'], correction_method=':
```

```
Out[29]:    Independent Variable Independent Variable Value 1  \
        0              Technique                     PhoneOnly
        1              Technique                     PhoneOnly
        2              Technique                  PhoneInArOut
        3              Technique                     PhoneOnly
        4              Technique                     PhoneOnly
        5              Technique                  PhoneInArOut
        6              Technique                     PhoneOnly
        7              Technique                     PhoneOnly
        8              Technique                  PhoneInArOut
        9              Technique                     PhoneOnly
        10             Technique                     PhoneOnly
        11             Technique                  PhoneInArOut
        12             Technique                     PhoneOnly
        13             Technique                     PhoneOnly
        14             Technique                  PhoneInArOut

           Independent Variable Value 2      Dependent Variable  Mann-Whitney U  \
        0                  PhoneInArOut      Easy to Understand            78.0
        1                 MidAirInArOut      Easy to Understand            45.5
        2                 MidAirInArOut      Easy to Understand            50.5
        3                  PhoneInArOut    Mentally Easy to Use            24.0
        4                 MidAirInArOut    Mentally Easy to Use            33.0
        5                 MidAirInArOut    Mentally Easy to Use            81.5
        6                  PhoneInArOut        Subjective Speed            78.0
        7                 MidAirInArOut        Subjective Speed            55.5
        8                 MidAirInArOut        Subjective Speed            25.5
        9                  PhoneInArOut  Subjective Performance            23.5
        10                MidAirInArOut  Subjective Performance            72.5
        11                MidAirInArOut  Subjective Performance            20.0
        12                 PhoneInArOut              Preference            16.5
        13                MidAirInArOut              Preference            61.5
        14                MidAirInArOut              Preference            35.5

            p-value
        0   0.222479
        1   0.006894
        2   0.022714
        3   0.002369
        4   0.006894
        5   0.441079
        6   0.395305
        7   0.099006
        8   0.002509
        9   0.002360
        10  0.305657
        11  0.002007
        12  0.001769
        13  0.140205
        14  0.006894
```

Significant results are:

- Easy to Understand : *PhoneOnly* (5.00±0.00) is significantly better than *MidAirInArOut* (4.31±0.82);
- Mentally Easy to Use : *PhoneOnly* (3.31±0.82) is significantly worst than *PhoneInArOut* (4.54±0.63) and *MidAirInArOut* (4.38±0.92);
- Subjective Speed : *PhoneInArOut* (4.15±1.17) is significantly better than *MidAirInArOut* (2.85±1.29);
- Subjective Performance : *PhoneInArOut* (4.38±0.62) is significantly better than *PhoneOnly* (3.46±1.01) and *MidAirInArOut* (3.23±1.05);
- Frustration : *PhoneInArOut* (4.08±1.14) is significantly better than *PhoneOnly* (3.46±1.55) and *MidAirInArOut* (2.69±1.26);
- Preference: *PhoneInArOut* (1.31±0.46) is significantly prefered to *PhoneOnly* (2.54±0.63) or *MidAirInArOut* (2.15±0.77).

Display the mean of the ranks with the standard deviation for each question among TECHNIQUE:

In [30]:
```
ranks.groupby([trials_ivs['technique']['label']])\
      .aggregate(lambda x : '{:.2f}±{:.2f}'.format(np.mean(x), np.std(x)))\
      .loc[:, ranks_dvs.loc['label', :]]
```

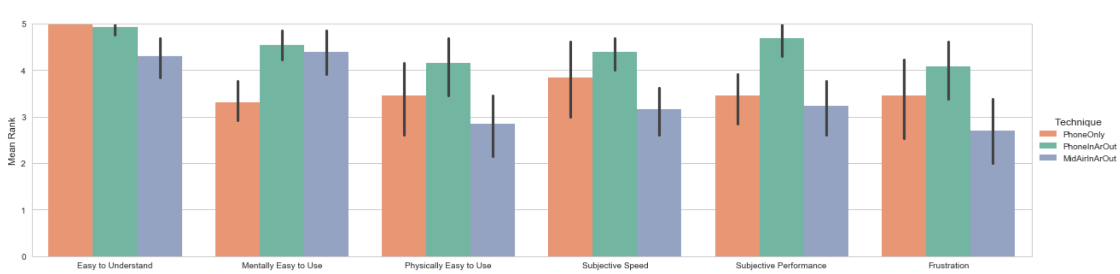
Out[30]:

Technique	Easy to Understand	Mentally Easy to Use	Physically Easy to Use
PhoneOnly	5.00±0.00	3.31±0.82	3.46±1.45
PhoneInArOut	4.92±0.27	4.54±0.63	4.15±1.17
MidAirInArOut	4.31±0.82	4.38±0.92	2.85±1.29

Technique	Subjective Speed	Subjective Performance	Frustration	Preference
PhoneOnly	3.85±1.46	3.46±1.01	3.46±1.55	2.54±0.63
PhoneInArOut	4.38±0.62	4.69±0.61	4.08±1.14	1.31±0.46
MidAirInArOut	3.15±0.95	3.23±1.05	2.69±1.26	2.15±0.77

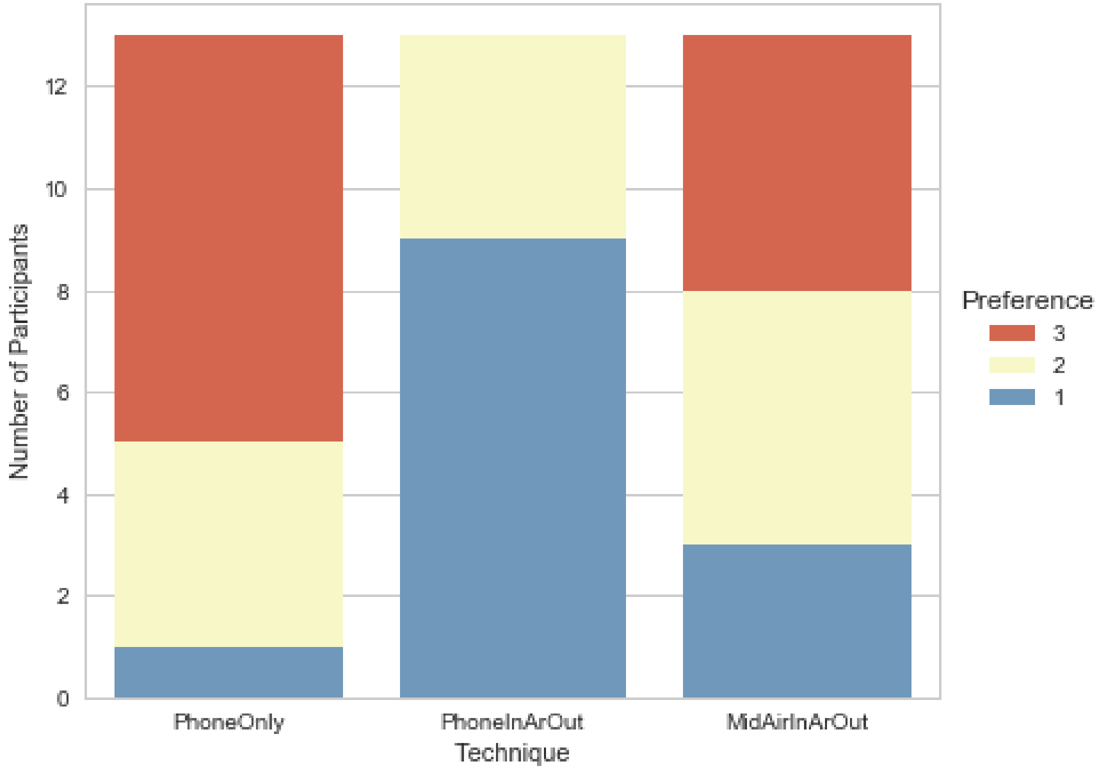
In [31]:
```
plt.figure(figsize=(21,5))

melted_ranks = pd.melt(ranks, id_vars=[ordering['label'], technique['label']], var_name=labels
                       value_name=labels['mean_rank'], value_vars=ranks_dvs.loc['label', 'easy_

ax = sns.barplot(x=labels['question'], y=labels['mean_rank'], hue=technique['label'], palette=
                 data=melted_ranks)
ax.set(ylim=(0, ranks_dvs['frustration']['scale'][-1]))
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5), title=technique['label'])
```

Out[31]: <matplotlib.legend.Legend at 0x29f35e2f710>



In [32]:
```
(fig, axs) = plot_ranks_distributions('technique', ['preference'])
axs[0].legend(loc='center left', bbox_to_anchor=(1, 0.5), title=ranks_dvs['preference']['label
```

Out[32]:
```
<matplotlib.legend.Legend at 0x29f35da52e8>
```



3 3. Participant rates

Some functions for the analysis:

In [33]:
```
def melt_trials(value_vars, var_name, value_name, data=trials):
    return pd.melt(data, id_vars=trials_ivs.loc['label', :], value_vars=value_vars, var_name=va
```

In [34]:
```
def filter_trials_outliers(dv_index, data=trials, nth_percentile_trimed=5):
    dv = trials_dvs[dv_index]
    pmin, pmax = np.nanpercentile(data[dv], [nth_percentile_trimed, 100 - nth_percentile_trimed
    return data[(np.isnan(data[dv]) == False) & (pmin < data[dv]) & (data[dv] < pmax)]
```

In [35]:
```
def test_pairwise_trials(dv_index, iv_index, data=trials, **args):
    results = []
    iv, dv = trials_ivs[iv_index], trials_dvs[dv_index]
    iv_category_ids = range(len(iv['categorical']))
```

```
            samples = [data[data[iv['label']] == iv_value][dv] for iv_value in iv['categorical']]
            for iv_value_ids in itertools.combinations(iv_category_ids, 2):
                T, p_value = stats.ttest_ind(samples[iv_value_ids[0]], samples[iv_value_ids[1]])

                mean_diff = np.mean(samples[iv_value_ids[0]]) - np.mean(samples[iv_value_ids[1]])
                mean_diff_percentage = mean_diff / np.mean(samples[iv_value_ids[1]]) * 100

                results.append([iv['label'], iv['categorical'][iv_value_ids[0]], iv['categorical'][iv_v
                                dv, mean_diff, mean_diff_percentage, T, p_value])

        results = pd.DataFrame(results, columns=['Independent Variable', 'Independent Variable Valu
                                                 'Independent Variable Value 2', 'Dependent Variabl
                                                 'Mean Difference', 'Mean Difference Percentage',\
                                                 'T statistic', 'p-value'])

        p_values_correction(results, **args)
        return results
```

```
In [36]: def test_pairwise_trial_conditions(dv_index, iv_index, condition_iv_indexes, data=trials, **arg
             condition_ivs = [trials_ivs[iv_index] for iv_index in condition_iv_indexes]
             condition_iv_cats = [iv['categorical'] for iv in condition_ivs]

             results_list = []
             for condition_iv_values in itertools.product(*condition_iv_cats):
                 sample = data
                 for condition_iv, condition_iv_value in zip(condition_ivs, condition_iv_values):
                     sample = sample[sample[condition_iv['label']] == condition_iv_value]

                 r = test_pairwise_trials(dv_index, iv_index, data=sample, correction_method=None)
                 r['Condition'] = [condition_iv_values] * len(r)
                 results_list.append(r)

             results = results_list[0]
             for result in results_list[1:]:
                 results = results.append(result)

             p_values_correction(results, **args)
             results.reset_index(inplace=True, drop=True)
             return results
```

```
In [37]: def test_non_normal_trials(dv_indexes, iv_indexes, data=trials, **args):
             results = []

             for dv_index in dv_indexes:
                 dv = trials_dvs[dv_index]

                 for iv_index in iv_indexes:
                     iv = trials_ivs[iv_index]
                     samples = [data[data[iv['label']] == iv_value][dv] for iv_value in iv['categorical

                     kruskal_H, p_value = stats.kruskal(*samples)
                     results.append([iv['label'], dv, kruskal_H, p_value])
```

```
        results = pd.DataFrame(results, columns=['Independent Variable', 'Dependent Variable',\
                                                 'Kruskal-Wallis H', 'p-value'])
        p_values_correction(results, **args)
        return results
```

```
In [38]: def test_pairwise_non_normal_trials(dv_indexes, iv_indexes, data=trials, **args):
        results = []

        for dv_index in dv_indexes:
            dv = trials_dvs[dv_index]

            for iv_index in iv_indexes:
                iv = trials_ivs[iv_index]
                samples = [data[data[iv['label']] == iv_value][dv] for iv_value in iv['categorical

                iv_category_ids = range(len(iv['categorical']))
                for iv_value_ids in itertools.combinations(iv_category_ids, 2): # Test each pair
                    U, p_value = stats.mannwhitneyu(samples[iv_value_ids[0]], samples[iv_value_ids
                    results.append([iv['label'], iv['categorical'][iv_value_ids[0]], iv['categorica
                                    dv, U, p_value])

        results = pd.DataFrame(results, columns=['Independent Variable', 'Independent Variable Valu
                                                 'Independent Variable Value 2', 'Dependent Variabl
                                                 'Mann-Whitney U', 'p-value'])
        p_values_correction(results, **args)
        return results
```

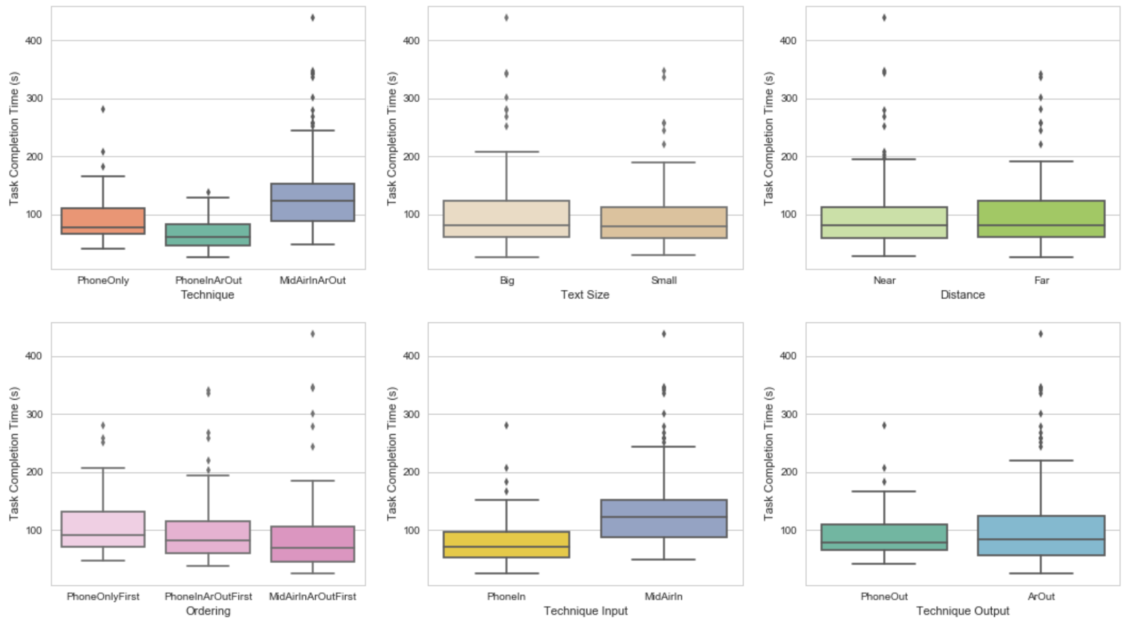
```
In [39]: def plot_trials(dv_index, iv_indexes_list=[], data=trials, kind='bar'):
        dv = trials_dvs[dv_index]

        if (len(iv_indexes_list) == 0):
            iv_indexes_list = [[iv_index] for iv_index in trials_ivs.columns]

        fig, axs = subplots(len(iv_indexes_list))

        for iv_indexes, ax in zip(iv_indexes_list, axs):
            ivs = [trials_ivs[iv_index] for iv_index in iv_indexes]

            if (len(ivs) == 1):
                iv = ivs[0]

                if (kind == 'bar'):
                    sns.barplot(x=iv['label'], y=dv, data=data, palette=iv['palette'], ax=ax)

                elif (kind == 'box'):
                    sns.boxplot(x=iv['label'], y=dv, data=data, palette=iv['palette'], ax=ax)

                elif (kind == 'count'):
                    sns.countplot(hue=iv['label'], x=dv, data=data, palette=iv['palette'], ax=ax)
                    ax.set(ylabel='Count')
                    ax.legend(loc='upper right', title=labels['count'], frameon=True)

            elif (len(ivs) == 2):
                if (kind == 'bar'):
```

```
                sns.barplot(x=ivs[1]['label'], y=dv, hue=ivs[0]['label'], data=data, palette=i
            ax.legend(frameon=True, loc='upper left', bbox_to_anchor=(1, 1))

    return (fig, axs)
```

3.1 3.1. Task completion time

Visualize the TCT distributions for each IV.

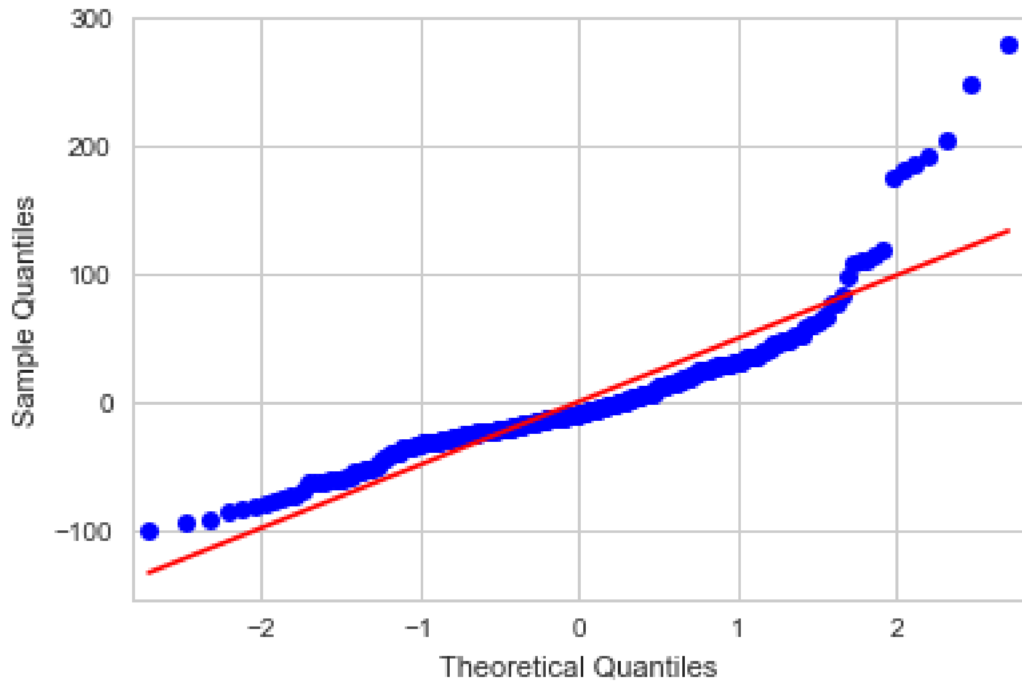
```
In [40]: (fig, axs) = plot_trials('total_time', kind='box')
```



We should test all the assumptions of an ANOVA (independence of measure points, normality, homogeneity of variance). The trials are independent from each other, since they are generated randomly. The data is non-normal, but the ANOVA can tolerate non-normal data with skewed distribution. The homogeneity of variance is less important when the sample sizes are equal.

We perform a full factorial ANOVA with the model: TCT ~ TECHNIQUE x TEXT_SIZE x DISTANCE + ORDERING.

```
In [41]: tct_model = ols('total_time ~ technique * text_size * distance + ordering', data=trials_for_an
         tct_model_anova = sm.stats.anova_lm(tct_model, typ=2)

         sm.qqplot(tct_model.resid, line='s')
         plt.show()

         tct_model_anova
```




```
Out[41]:                              sum_sq     df          F        PR(>F)
         technique              256284.863291    2.0  49.887481  3.343713e-19
         text_size                6524.396184    1.0   2.540031  1.121455e-01
         distance                  273.284406    1.0   0.106393  7.445375e-01
         ordering                12606.272262    2.0   2.453891  8.784569e-02
         technique:text_size     10779.057863    2.0   2.098212  1.246423e-01
         technique:distance       1129.444917    2.0   0.219854  8.027777e-01
         text_size:distance      10418.141005    1.0   4.055915  4.499418e-02
         technique:text_size:distance  17350.562563    2.0   3.377397  3.556339e-02
         Residual               703804.357932  274.0        NaN           NaN
```

The main significant effect on TCT is TECHNIQUE (p<0.0001). There is also interaction effects: TEXT_SIZE x DISTANCE (p=0.046) and TECHNIQUE x TEXT_SIZE x DISTANCE (p=0.037). TEXT_SIZE, DISTANCE and ORDERING have no significant effects on TCT.

Also, we verify that the ANOVA residuals are roughly normal with the QQ-plot above.

We compare TCT for the three TECHNIQUE with pairwise t-tests (Benjamini–Hochberg correction) first. Then we compare TCT for the three TECHNIQUE for all TEXT_SIZE x DISTANCE conditions with pairwise t-tests (Benjamini–Hochberg correction).

```
In [42]: results = test_pairwise_trials('total_time', 'technique', correction_method=None)
         results['Condition'] = [''] * len(results)

         r = test_pairwise_trial_conditions('total_time', 'technique', ['text_size', 'distance'], corre
         results = results.append(r)
         results.reset_index(inplace=True, drop=True)

         p_values_correction(results, correction_method='fdr_bh')
         results
```

Out[42]:

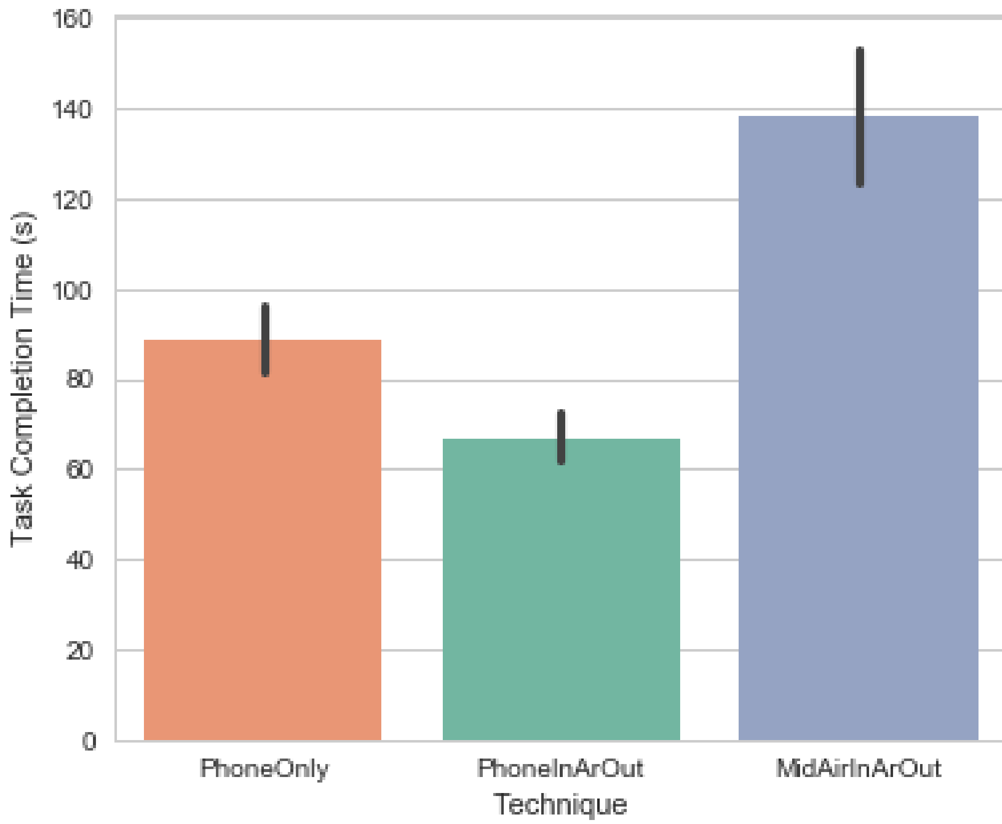
	Independent Variable	Independent Variable Value 1	Independent Variable Value 2	Dependent Variable	Mean Difference	Mean Difference Percentage	T statistic	p-value	Condition
0	Technique	PhoneOnly	PhoneInArOut	Task Completion Time (s)	21.840362	32.666819	4.600061	2.887821e-05	
1	Technique	PhoneOnly	MidAirInArOut	Task Completion Time (s)	-49.467731	-35.803106	-5.669051	3.951549e-07	
2	Technique	PhoneInArOut	MidAirInArOut	Task Completion Time (s)	-71.308093	-51.610437	-8.608496	4.178800e-14	
3	Technique	PhoneOnly	PhoneInArOut	Task Completion Time (s)	24.168476	36.724278	2.549015	1.936788e-02	(Big, Near)
4	Technique	PhoneOnly	MidAirInArOut	Task Completion Time (s)	-77.353492	-46.227392	-3.835002	8.149009e-04	(Big, Near)
5	Technique	PhoneInArOut	MidAirInArOut	Task Completion Time (s)	-101.521968	-60.670769	-5.199072	2.246879e-05	(Big, Near)
6	Technique	PhoneOnly	PhoneInArOut	Task Completion Time (s)	26.892856	41.152781	2.379929	2.482784e-02	(Big, Far)
7	Technique	PhoneOnly	MidAirInArOut	Task Completion Time (s)	-43.047900	-31.819082	-2.450544	2.265802e-02	(Big, Far)
8	Technique	PhoneInArOut	MidAirInArOut	Task Completion Time (s)	-69.940756	-51.697078	-4.614999	7.898347e-05	(Big, Far)
9	Technique	PhoneOnly	PhoneInArOut	Task Completion Time (s)	9.832919	13.926232	1.170723	2.477394e-01	(Small, Near)
10	Technique	PhoneOnly	MidAirInArOut	Task Completion Time (s)	-26.990175	-25.123437	-2.028633	5.175809e-02	(Small, Near)
11	Technique	PhoneInArOut	MidAirInArOut	Task Completion Time (s)	-36.823094	-34.276275	-2.721200	1.372939e-02	(Small, Near)
12	Technique	PhoneOnly	PhoneInArOut	Task Completion Time (s)	26.467198	40.306314	2.970482	7.855825e-03	(Small, Far)
13	Technique	PhoneOnly	MidAirInArOut	Task Completion Time (s)	-50.479357	-35.396365	-3.005064	7.855825e-03	(Small, Far)
14	Technique	PhoneInArOut	MidAirInArOut	Task Completion Time (s)	-76.946555	-53.955290	-4.776990	5.551087e-05	(Small, Far)

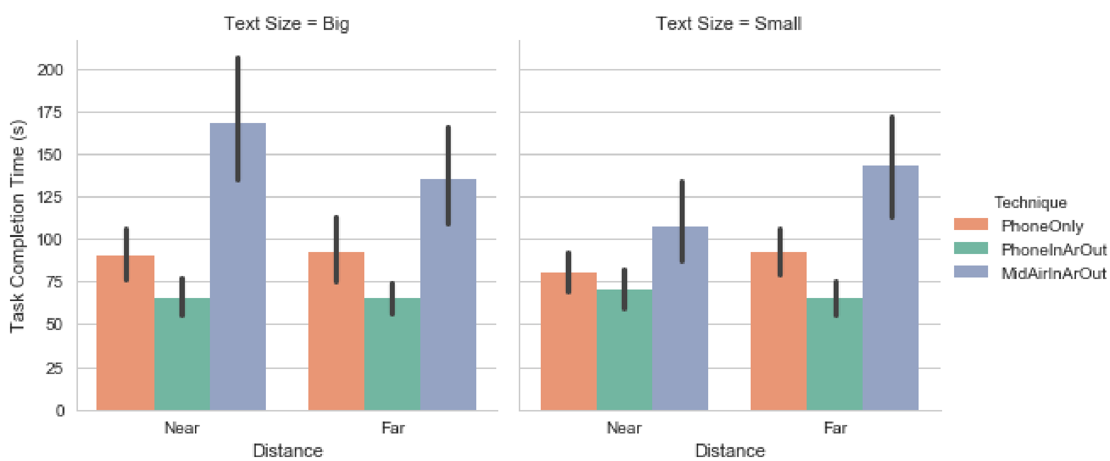
PhoneInArOut is 22 s faster than *PhoneOnly* ($p < 0.0001$, 33% faster) and 71 s faster than *MidAirInArOut* ($p < 0.0001$, 52% faster) faster than *MidAirInArOut*. Also, *PhoneOnly* is 50 s faster than *MidAirInArOut* ($p < 0.0001$, 36% faster). I hypothesed this TCT difference order between the three techniques.

However this is not the case for all TEXT_SIZE x DISTANCE condition:

- For *Big* text size and *Near* distance:
  - *PhoneInArOut* is 24 s faster than *PhoneOnly* ($p < 0.019$, 37% faster);
  - *PhoneInArOut* is 102 s faster than *MidAirInArOut* ($p < 0.0001$, 61% faster);
  - *PhoneOnly* is 77 s faster than *MidAirInArOut* ($p = 0.0008$, 46% faster).
- For *Big* text size and *Far* distance:
  - *PhoneInArOut* is 27 s faster than *PhoneOnly* ($p < 0.022$, 41% faster);
  - *PhoneInArOut* is 70 s faster than *MidAirInArOut* ($p < 0.0001$, 52% faster);
  - *PhoneOnly* is 43 s faster than *MidAirInArOut* ($p = 0.023$, 32% faster).
- For *Small* text size and *Near* distance:
  - There is no significant difference between *PhoneInArOut* and *PhoneOnly*;
  - *PhoneInArOut* is 70 s faster than *MidAirInArOut* ($p = 0.0003$, 52% faster);
  - There is no significant difference between nor *PhoneOnly* and *MidAirInArOut*.
- For *Small* text size and *Far* distance:
  - *PhoneInArOut* is 26 s faster than *PhoneOnly* ($p < 0.0079$, 40% faster);
  - *PhoneInArOut* is 77 s faster than *MidAirInArOut* ($p < 0.0001$, 54% faster);
  - *PhoneOnly* is 50 s faster than *MidAirInArOut* ($p = 0.0079$, 35% faster).

```
In [43]: (fig, axs) = plot_trials('total_time', [['technique']])

         sns.factorplot(x=distance['label'], y=trials_dvs['total_time'], col=text_size['label'], hue=te
                        data=trials, palette=technique['palette'], kind='bar')

Out[43]: <seaborn.axisgrid.FacetGrid at 0x29f35cdde48>
```



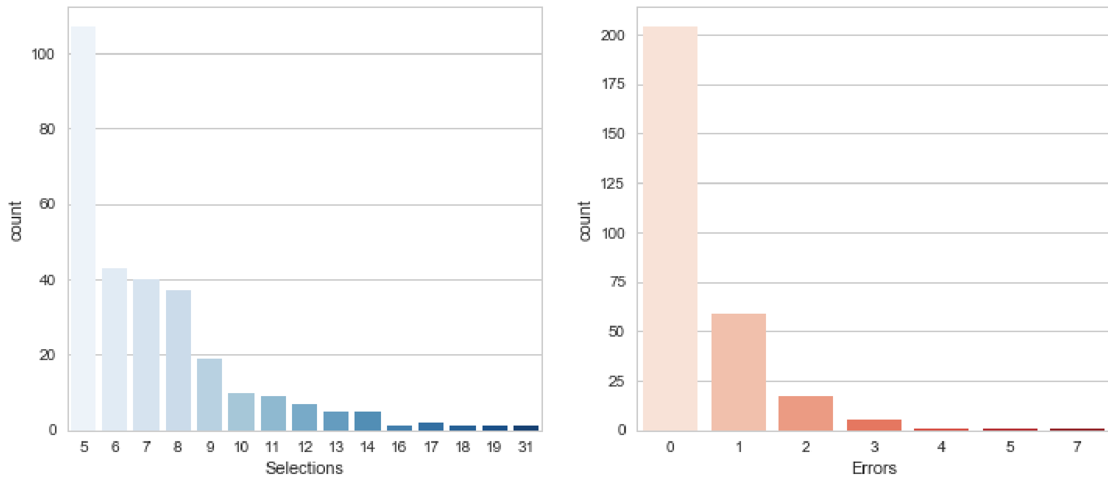


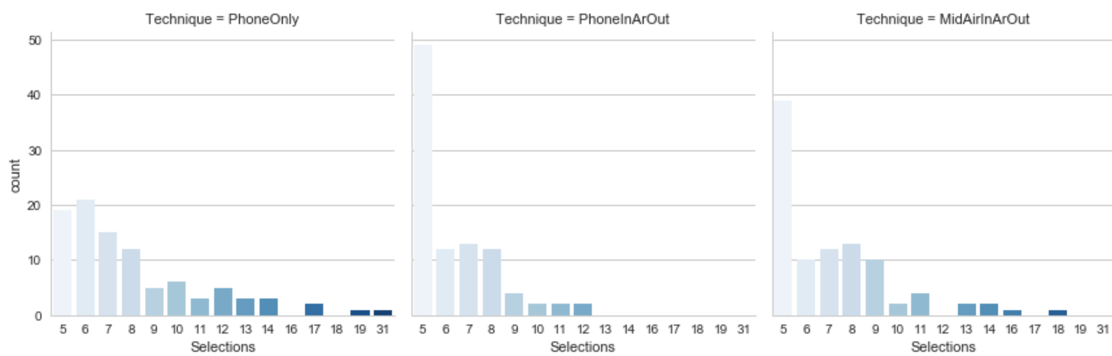
3.2 3.2. Error rate

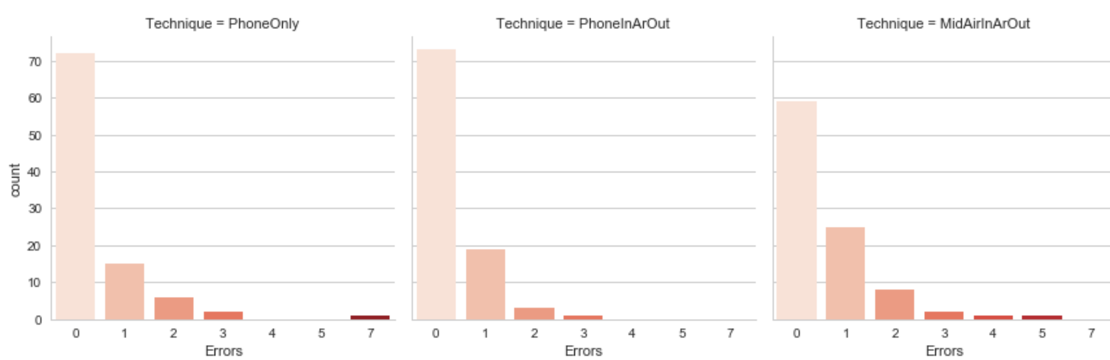
Visualize the SELECTIONS and ERRORS distributions:

```
In [44]: (fig, axs) = subplots(2)

         sns.countplot(x=trials_dvs['selections_count'], data=trials, palette='Blues', ax=axs[0])
         sns.countplot(x=trials_dvs['errors'], data=trials, palette='Reds', ax=axs[1])

         sns.factorplot(x=trials_dvs['selections_count'], col=technique['label'], kind='count', palette=
         sns.factorplot(x=trials_dvs['errors'], col=technique['label'], kind='count', palette='Reds', da
```

```
Out[44]: <seaborn.axisgrid.FacetGrid at 0x29f35e73e80>
```







We can't use ANOVA as for both SELECTIONS and ERRORS variables as their distributions are exponentials. We use Kruskal-Wallis test (Benjamini–Hochberg correction) on SELECTIONS and ERRORS to check if there is significative differences among TECHNIQUE, TEXT_SIZE, DISTANCE or ORDERING.

```
In [45]: test_non_normal_trials(['selections_count', 'errors'], ['technique', 'text_size', 'distance',
```

```
Out[45]:   Independent Variable Dependent Variable  Kruskal-Wallis H   p-value
         0            Technique         Selections         20.015292  0.000217
         1            Text Size         Selections          0.000810  0.977297
         2             Distance         Selections          0.329777  0.754387
         3             Ordering         Selections         19.648369  0.000217
         4            Technique             Errors          6.510257  0.077152
         5            Text Size             Errors          0.063506  0.915472
         6             Distance             Errors          0.437007  0.754387
         7             Ordering             Errors         10.200291  0.016256
```

Only TECHNIQUE (p=0.0002) and ORDERING (p=0.0002) have a significant effect on SELECTIONS. Identically, only TECHNIQUE (p=0.077) and ORDERING (p=0.016) have a significant effect on ERRORS.

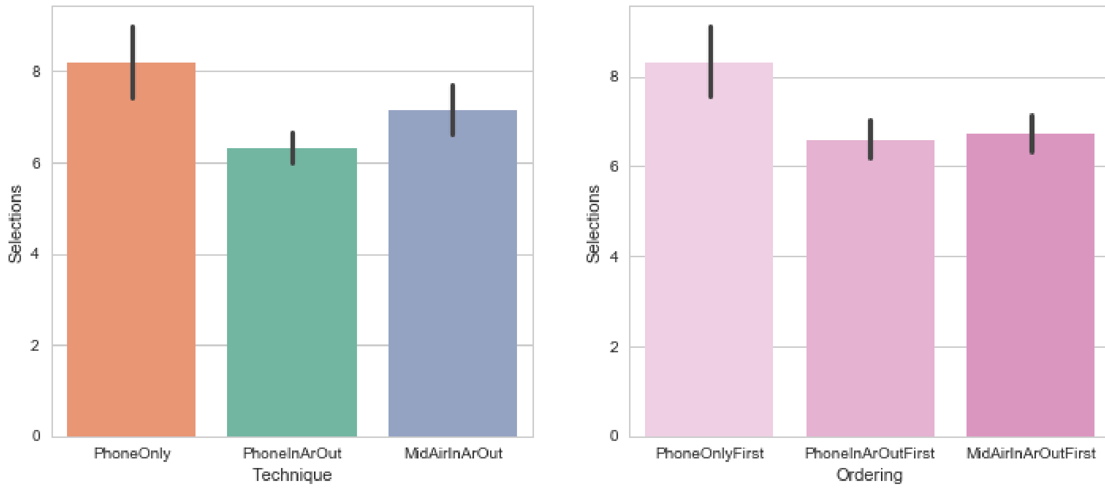
We compare SELECTIONS and ERRORS among TECHNIQUE with pairwise Mann-Whitney tests (Benjamini–Hochberg correction):

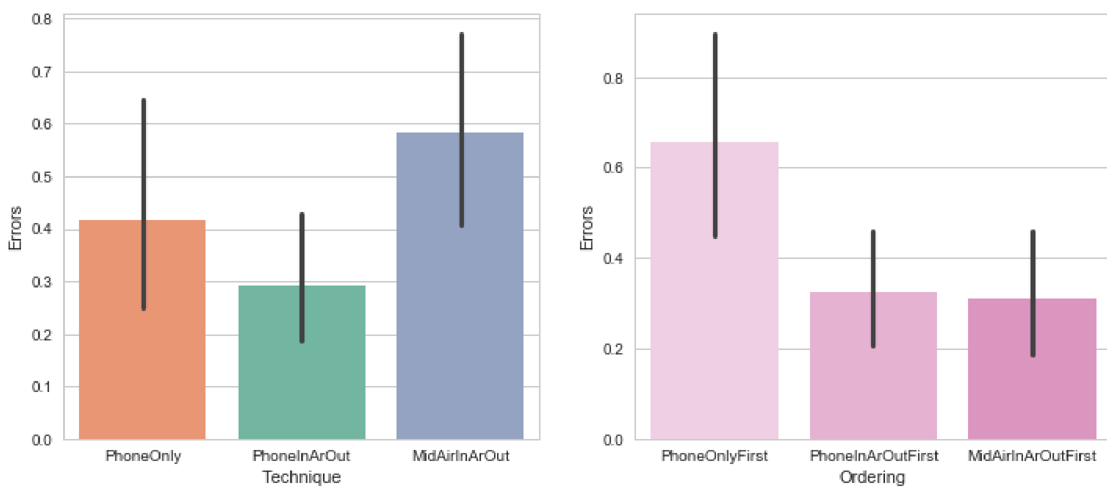
```
In [46]: test_pairwise_non_normal_trials(['selections_count', 'errors'], ['technique'])
```

```
Out[46]:   Independent Variable Independent Variable Value 1  \
         0            Technique                    PhoneOnly
         1            Technique                    PhoneOnly
         2            Technique                 PhoneInArOut
         3            Technique                    PhoneOnly
         4            Technique                    PhoneOnly
         5            Technique                 PhoneInArOut

           Independent Variable Value 2 Dependent Variable  Mann-Whitney U   p-value
         0                 PhoneInArOut         Selections          2923.0  0.000020
         1                MidAirInArOut         Selections          3733.5  0.020678
         2                MidAirInArOut         Selections          3850.5  0.028448
         3                 PhoneInArOut             Errors          4502.5  0.358201
         4                MidAirInArOut             Errors          4006.5  0.034230
         5                MidAirInArOut             Errors          3869.0  0.020678
```

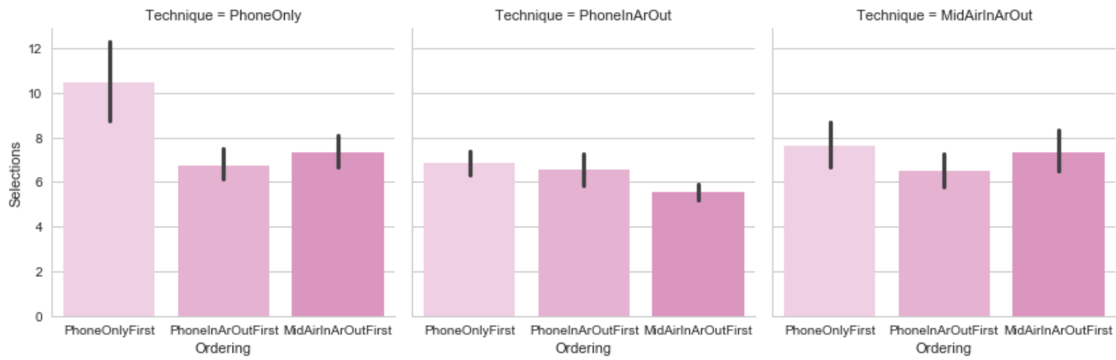
- Participants made a significant different number of selections between the three TECHNIQUE : *PhoneOnly* yielded the most selections (p=0.021) and *PhoneInArOut* the least selections (p=0.028). When using *PhoneOnly*, we observed that users sometimes forgot what they had selected or changed their mind during the drop operation, increasing the number of selections.
- Participants made significant more errors with *MidAirInArOut* rather than *PhoneOnly* (p=0.034) or *PhoneInArOut* (p=0.021), but these latter two do not differ significantly. With *MidAirInArOut*, users sometimes dropped items in the wrong container, not voluntarily but because they were too zoomed out and/or the sensing limitations of the Leap Motion made it difficult to successfully aim at targets, especially when arms were crossed.

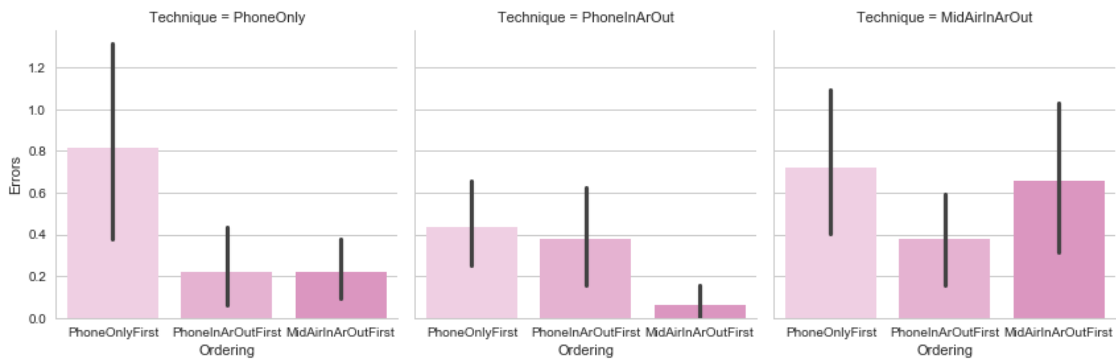
```
In [47]: (fig, axs) = plot_trials('selections_count', [['technique'], ['ordering']])
         (fig, axs) = plot_trials('errors', [['technique'], ['ordering']])
```






```
In [48]: sns.factorplot(x=ordering['label'], y=trials_dvs['selections_count'], col=technique['label'],\
                        palette=ordering['palette'], kind='bar', data=trials)

         sns.factorplot(x=ordering['label'], y=trials_dvs['errors'], col=technique['label'],\
                        palette=ordering['palette'], kind='bar', data=trials)

Out[48]: <seaborn.axisgrid.FacetGrid at 0x29f367fe828>
```

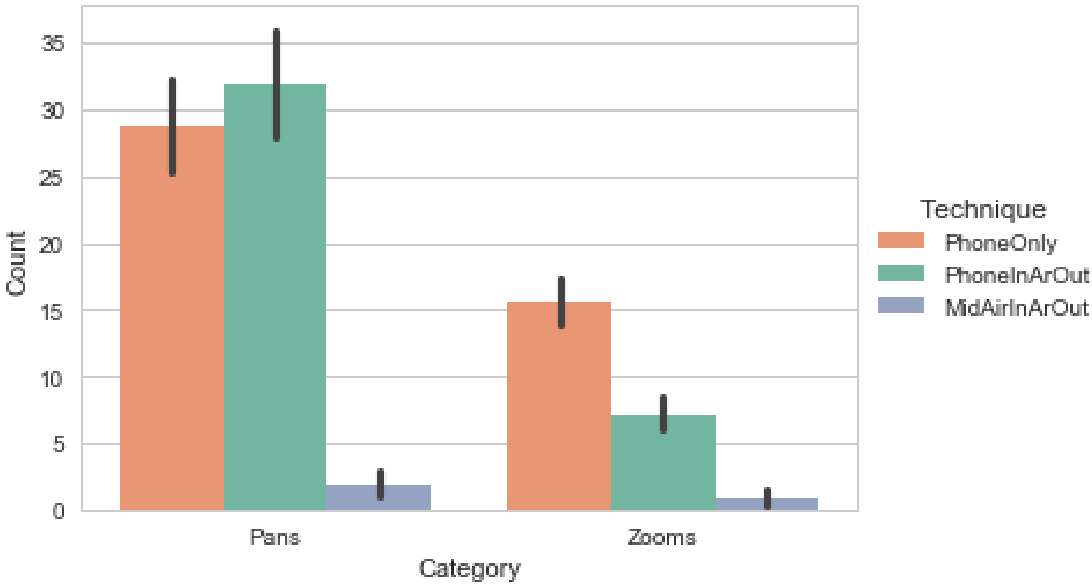




PhoneOnly seems to be more sensitive in terms of ERRORS to being the first technique tested by users.

3.3 3.3. Navigation

```
In [49]: trial_counts = melt_trials(var_name=labels['category'], value_name=labels['count'],\
                                    value_vars=[trials_dvs['pan_count'], trials_dvs['zoom_count']])

         plt.figure(figsize=(6,4))
         ax = sns.barplot(x=labels['category'], y=labels['count'], hue=technique['label'],\
                          palette=technique['palette'], data=trial_counts)
         ax.legend(loc='center left', bbox_to_anchor=(1, 0.5), title=technique['label'])
         plt.show()

         trial_counts.groupby([technique['label'], labels['category']], sort=False)\
                     .aggregate(lambda x : '{:.2f}±{:.2f}'.format(np.mean(x), np.std(x)))\
                     .unstack(level=1)
```

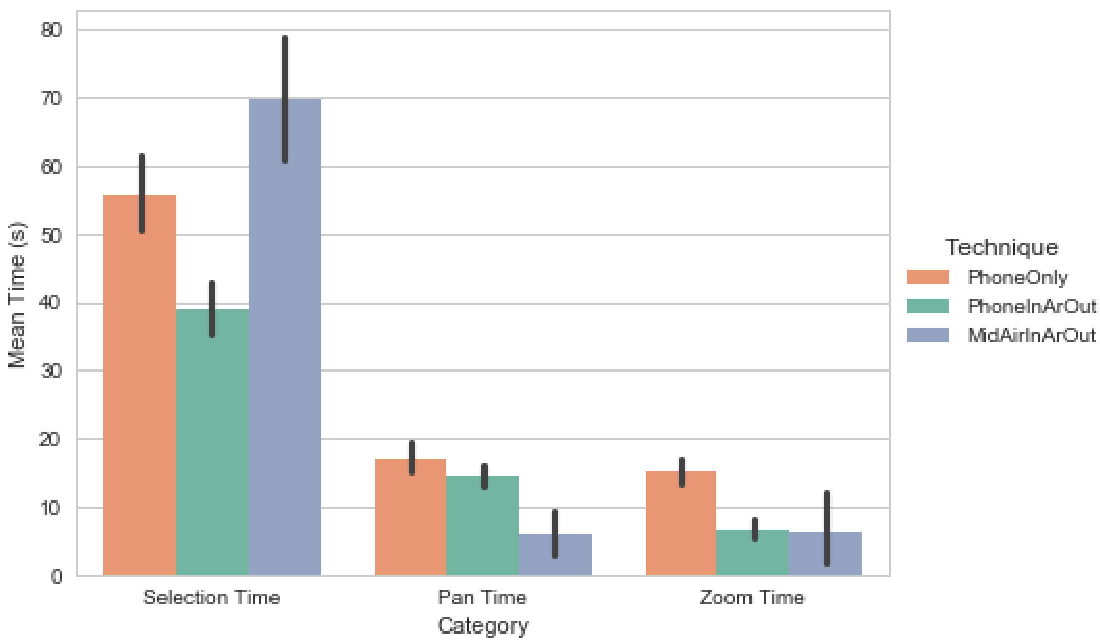


Out[49]:

```
                         Count
Category                  Pans        Zooms
Technique
PhoneOnly          28.70±18.56   15.53±8.63
PhoneInArOut       31.81±20.82    7.14±6.19
MidAirInArOut       1.83±4.79     0.83±3.20
```

In [50]:

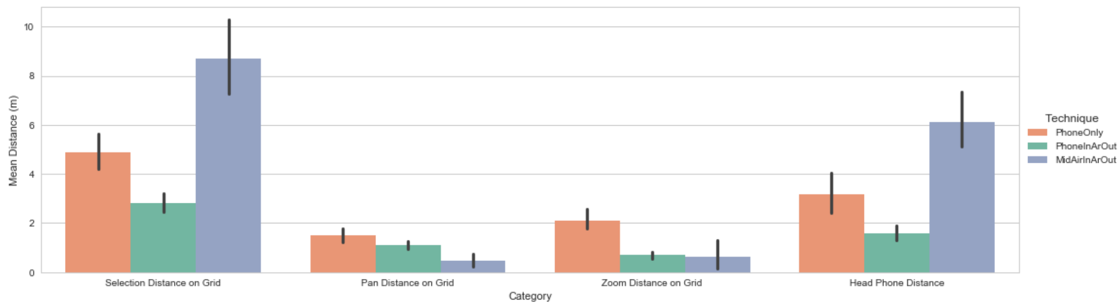
```
trial_times = melt_trials(var_name=labels['category'], value_name=labels['time'],\
                          value_vars=[trials_dvs['selections_time'], trials_dvs['pan_time'], t

plt.figure(figsize=(7,5))
ax = sns.barplot(x=labels['category'], y=labels['time'], hue=technique['label'], palette=techn
                 data=trial_times)
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5), title=technique['label'])
plt.show()

trial_times.groupby([technique['label'], labels['category']], sort=False)\
            .aggregate(lambda x : '{:.2f}±{:.2f}'.format(np.mean(x), np.std(x)))\
            .unstack(level=1)
```



Out[50]:

Category	Mean Time (s) Selection Time	Pan Time	Zoom Time
Technique			
PhoneOnly	55.74±27.54	17.12±10.66	15.18±9.00
PhoneInArOut	38.98±19.21	14.53±8.10	6.70±6.83
MidAirInArOut	69.68±47.17	5.97±16.56	6.24±26.31

In [51]:
```
trial_distances = melt_trials(var_name=labels['category'], value_name=labels['distance'],\
                              value_vars=[trials_dvs['selections_projected_distance'],\
                                          trials_dvs['pan_projected_distance'],\
                                          trials_dvs['zoom_projected_distance'],\
                                          trials_dvs['absolute_head_phone_distance']])

plt.figure(figsize=(18,5))
ax = sns.barplot(x=labels['category'], y=labels['distance'], hue=technique['label'], palette=te
                 data=trial_distances)
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5), title=technique['label'])
plt.show()

trial_distances.groupby([technique['label'], labels['category']], sort=False)\
               .aggregate(lambda x : '{:.2f}±{:.2f}'.format(np.mean(x), np.std(x)))\
               .unstack(level=1)
```




```
Out[51]:                            Mean Distance (m)                         \
Category      Selection Distance on Grid Pan Distance on Grid
Technique
PhoneOnly                      4.87±3.74            1.50±1.31
PhoneInArOut                   2.83±1.89            1.09±0.76
MidAirInArOut                  8.72±7.89            0.47±1.30


Category      Zoom Distance on Grid Head Phone Distance
Technique
PhoneOnly                 2.11±1.99           3.18±3.55
PhoneInArOut              0.69±0.74           1.57±1.37
MidAirInArOut             0.62±2.77           6.12±5.05
```

Variable meanings:

- Selection Time = time spent looking for where to drop an item that had been picked
- Selection Distance = distance travelled by the finger with an item selected
- Head Phone Distance = sum of the distance between the head and the phone

Results are:

- Both for *PhoneInArOut* and *PhoneOnly*, participants used pans more than zooms : in count, in time and in distance.
- Participants were the most effective in **selection time** for *PhoneInArOut* (38.98±19.21 s) rather than for *PhoneOnly* (55.74±27.54 s). We observed that the screen size in *PhoneInArOut* helped to make decisions on items to select or where to drop the selected items.
- Participants used as much **pans** in *PhoneInArOut* as in *PhoneOnly* (~30±19). But it seems they were slightly more effective with in *PhoneInArOut* rather than *PhoneOnly* both in time (14.53±8.10 s / 17.12±10.66 s) and distance (1.09±0.76 m / 1.50±1.31 m).
- Participants used less **zooms** in *PhoneInArOut* (7.14±6.19) rather than *PhoneOnly* (15.53±8.63). They were also more effective with zooms in *PhoneInArOut* rather than *PhoneOnly* both in time (6.70±6.83 s / 15.18±9.00 s) and distance (0.69±0.74 m / 2.11±1.99 m).
- In terms of **physical navigation**, the head-phone distance is the lowest for *PhoneInArOut* (1.57±1.37), the greatest for *MidAirInArOut* (6.12±5.05). *PhoneOnly* is between the two (3.18±3.55). In *MidAirInArOut*, participants moved in conjunction the hand and the phone to select the item: for items at the grid's extremities, it could be easier to rotate the phone to bring the item closer to the head. It seemed that people using both their hands were

more effective. Also, both for *PhoneOnly* and *MidAirInArOut*, participants preferred to bring closer the phone if they had trouble to read an item's letter. *PhoneInArOut* required less head-phone movement because participants could let the grid zoomed in and do only pans and drag'n'drop with items to complete the task without many virtual zoom nor physical zoom.