Handheld VESAD Analysis

March 20, 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
        import matplotlib.patches as patches

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

```
In [2]: %matplotlib inline

        # Style and size of the figures
        legend_fontsize = sns.plotting_context('notebook')['axes.labelsize']
        legend_title_fontsize = sns.plotting_context('notebook')['axes.titlesize']+1

        sns.set(context='notebook', style='whitegrid', font_scale=1.25,\
                rc={'legend.fontsize': legend_fontsize})
        subplotsize = (5,4)
```

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

Data are loaded in the following variables: - 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']
```

```python
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] # Pink pa
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] # Brown tr
trials_ivs.at['palette', 'input'] = [default_palette[5], default_palette[2]]
trials_ivs.at['palette', 'output'] = [default_palette[0], sns.color_palette('muted')[5]]
```

In [7]:
```python
# Trials DVs
if language == 'Français':
    trials_dv_labels = ['Temps de complétion (s)', 'Sélections', 'Temps en sélection',\
                         'Distance 3D en sélection', 'Distance en 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', 'Mouvements 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', 'Phone-Head Motion',\
                        'Signed Phone-Head Motion']

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

In [8]:
```python
# Ranks DVs
if language == 'Français':
    ranks_dv_labels = ['Facile à comprendre', 'Mentalement facile à utiliser',\
                        'Physiquement facile à utiliser', 'Rapidité', 'Réussite',\
                        '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(ranl
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)), ordere
ranks_dvs.at['palette', 'preference'] = [sns.color_palette('RdYlBu', 5)[i] for i in range(4,-1,
```

```
In [9]: # Shortcuts
        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
```

Clean the data:

```
In [10]: # 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 [11]: # Set the participant_id column as the index in participants
         participants.set_index(participants_ivs['participant_id'], inplace=True)
```

```
In [12]: # Some participants are non valid or don't have complete measures
         non_valid_participants = [0, 4]
         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 [13]: # 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 [14]: # 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']].

         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

         ranks[technique['label']] = ranks[technique['label']].astype(technique['categorical'])
```

```
    ranks[ordering['label']] = ranks[ordering['label']].astype(ordering['categorical'])

    # Rename categories
    if language == 'Français':
        technique['categorical'].categories = ['Téléphone', 'VESAD tactile', 'VESAD']
        text_size['categorical'].categories = ['Grand', 'Petit']
        distance['categorical'].categories = ['Proche', 'Loin']
        ordering['categorical'].categories = ['Groupe 1', 'Groupe 2', 'Groupe 3']
        iv_input['categorical'].categories = ['Tactile', 'Autour du téléphone']
        iv_output['categorical'].categories = ['Téléphone seul', 'Téléphone étendu']
    else:
        ordering['categorical'].categories = ['Group 1', 'Group 2', 'Group 3']

    # Set renamed categories to data
    participants[trials_ivs['ordering']['label']].cat.categories = ordering['categorical'].categori

    for iv_index in trials_ivs.columns:
        iv = trials_ivs[iv_index]
        trials[iv['label']].cat.categories = iv['categorical'].categories

    ranks[technique['label']].cat.categories = technique['categorical'].categories
    ranks[ordering['label']].cat.categories = ordering['categorical'].categories
```

```
In [15]: # 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 [16]: labels = pd.Series()

         if language == 'Français':
             labels['category'] = 'Catégorie'
             labels['count'] = 'Nombre total'
             labels['distance'] = 'Distance moyenne (m)'
             labels['dv'] = 'Variable dépendante'
             labels['iv'] = 'Variable indépendante'
             labels['iv_value'] = 'Valeur variable indépendante'
             labels['mean_difference'] = 'Différence des moyennes'
             labels['mean_difference_percentage'] = 'Différence des moyennes (%)'
             labels['mean_rank'] = 'Note moyenne'
             labels['preferences'] = ['Premier', 'Deuxième', 'Troisième']
             labels['p_value'] = 'Valeur p'
             labels['question'] = 'Question'
             labels['rank'] = 'Note'
             labels['time'] = 'Temps moyen (s)'
             labels['t_statistic'] = 'Statistique T'
             labels['votes'] = 'Votes'
         else:
             labels['category'] = 'Category'
             labels['count'] = 'Count'
             labels['distance'] = 'Mean Distance (m)'
```

```
        labels['dv'] = 'Dependent Variable'
        labels['iv'] = 'Independent Variable'
        labels['iv_value'] = 'Independent Variable Value'
        labels['mean_difference'] = 'Mean Difference'
        labels['mean_difference_percentage'] = 'Mean Difference Percentage'
        labels['mean_rank'] = 'Mean Rank'
        labels['preferences'] = ['First', 'Second', 'Third']
        labels['p_value'] = 'p-value'
        labels['question'] = 'Question'
        labels['rank'] = 'Rank'
        labels['time'] = 'Mean Time (s)'
        labels['t_statistic'] = 'T statistic'
        labels['votes'] = 'Votes'
```

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

         pd.DataFrame.mask = mask
```

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

```
In [19]: def subplots(nsubplots, ncols_max=2, subplotsize=subplotsize, *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]: def fix_legend_fontsize(ax_legend, fontsize=legend_title_fontsize):
             plt.setp(ax_legend.get_title(), fontsize=fontsize)
```

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(y=data.columns, x=row, label=row_index, color=palette[row_index-1], orient=
```

```
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(xlabel=labels['votes'], xlim=(0, cumulated_ranks_count.max()[0]))
                 ax.xaxis.set_major_locator(ticker.MultipleLocator(2))  # Fix the axis ticks

                 ax_handles, ax_labels = ax.get_legend_handles_labels()
                 legend = ax.legend(ax_handles[::-1], ax_labels[::-1], frameon=True, loc='lower center'
                                    bbox_to_anchor=(0.5, 1), mode=None, ncol=len(dv['scale']), title=dv
                                    fontsize=legend_fontsize-2)
                 fix_legend_fontsize(legend)

             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)
```

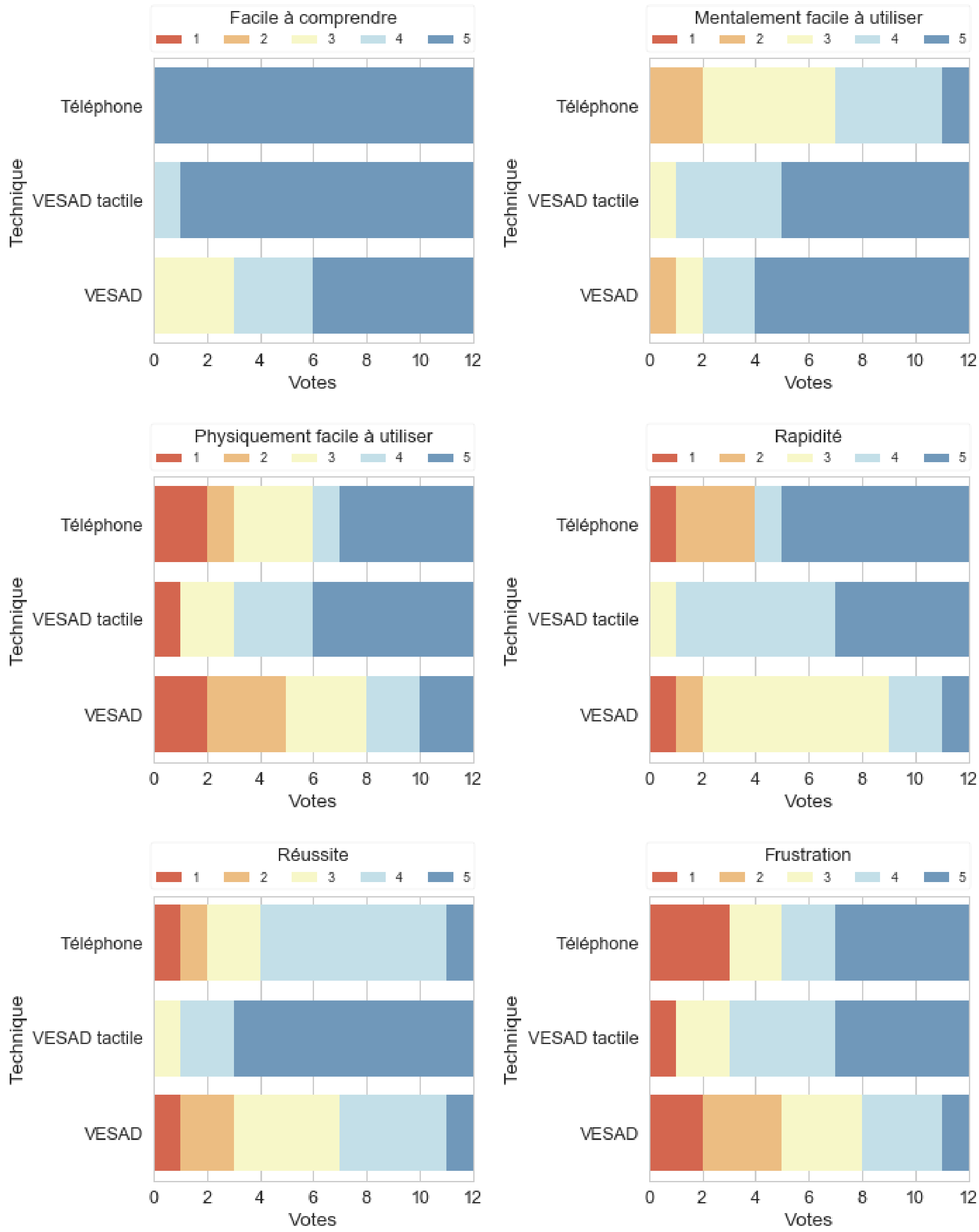
```
In [25]: def test_ranks(iv_index, dv_indexes, **args):
             iv = trials_ivs[iv_index]

             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=[labels['iv'], labels['dv'], 'Kruskal-Wallis H', la
        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=[labels['iv'], labels['iv_value'] + ' (1)',\
                                                      labels['iv_value'] + ' (2)', labels['dv'],\
                                                      'Mann-Whitney U', labels['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[0:-1])
         fig.savefig('ranks_distributions.png', dpi=300, bbox_inches='tight')
```



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]:   Variable indépendante             Variable dépendante  Kruskal-Wallis H  \
        0              Technique             Facile à comprendre         11.001420
```

```
1             Technique   Mentalement facile à utiliser         10.905742
2             Technique  Physiquement facile à utiliser          4.148121
3             Technique                        Rapidité          7.130846
4             Technique                        Réussite         13.784269
5             Technique                     Frustration          4.500057
6             Technique                      Préférence         12.638889

   Valeur p
0  0.007497
1  0.007497
2  0.125674
3  0.039599
4  0.006303
5  0.122962
6  0.006303
```

All the questions, except Physically Easy to Use and Frustration, 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), 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]:    Variable indépendante Valeur variable indépendante (1)  \
        0              Technique                        Téléphone
        1              Technique                        Téléphone
        2              Technique                     VESAD tactile
        3              Technique                        Téléphone
        4              Technique                        Téléphone
        5              Technique                     VESAD tactile
        6              Technique                        Téléphone
        7              Technique                        Téléphone
        8              Technique                     VESAD tactile
        9              Technique                        Téléphone
        10             Technique                        Téléphone
        11             Technique                     VESAD tactile
        12             Technique                        Téléphone
        13             Technique                        Téléphone
        14             Technique                     VESAD tactile

           Valeur variable indépendante (2)            Variable dépendante  \
        0                     VESAD tactile            Facile à comprendre
        1                             VESAD            Facile à comprendre
        2                             VESAD            Facile à comprendre
        3                     VESAD tactile  Mentalement facile à utiliser
        4                             VESAD  Mentalement facile à utiliser
        5                             VESAD  Mentalement facile à utiliser
        6                     VESAD tactile                       Rapidité
        7                             VESAD                       Rapidité
        8                             VESAD                       Rapidité
        9                     VESAD tactile                       Réussite
        10                            VESAD                       Réussite
```

```
11                              VESAD                       Réussite
12                      VESAD tactile                     Préférence
13                              VESAD                     Préférence
14                              VESAD                     Préférence

    Mann-Whitney U  Valeur p
0             66.0  0.215732
1             36.0  0.008605
2             40.5  0.020977
3             23.0  0.005031
4             28.5  0.010151
5             69.5  0.475027
6             70.5  0.475027
7             48.5  0.126838
8             21.0  0.004655
9             22.5  0.004655
10            57.0  0.215732
11            17.5  0.003904
12            16.0  0.003904
13            56.0  0.215732
14            32.0  0.013011
```

Significant results are:

- Easy to Understand : *PhoneOnly* (5.00±0.00) is significantly better than *MidAirInArOut* (4.25±0.83, p=0.009);
- Mentally Easy to Use : *PhoneOnly* (3.33±0.85) is significantly worst than *PhoneInArOut* (4.50±0.65, p=0.005) and *MidAirInArOut* (4.42±0.95, p=0.01);
- Subjective Speed : *PhoneInArOut* (4.33±0.62) is significantly better than *MidAirInArOut* (3.08±0.95, p=0.05);
- Subjective Performance : *PhoneInArOut* (4.67±0.62) is significantly better than *PhoneOnly* (3.50±1.04, p=0.005) and *MidAirInArOut* (3.17±1.07, p=0.004);
- Preference: *PhoneInArOut* (1.33±0.47) is significantly prefered to *PhoneOnly* (2.50±0.65, p=0.004) or *MidAirInArOut* (2.17±0.80, p=0.01).

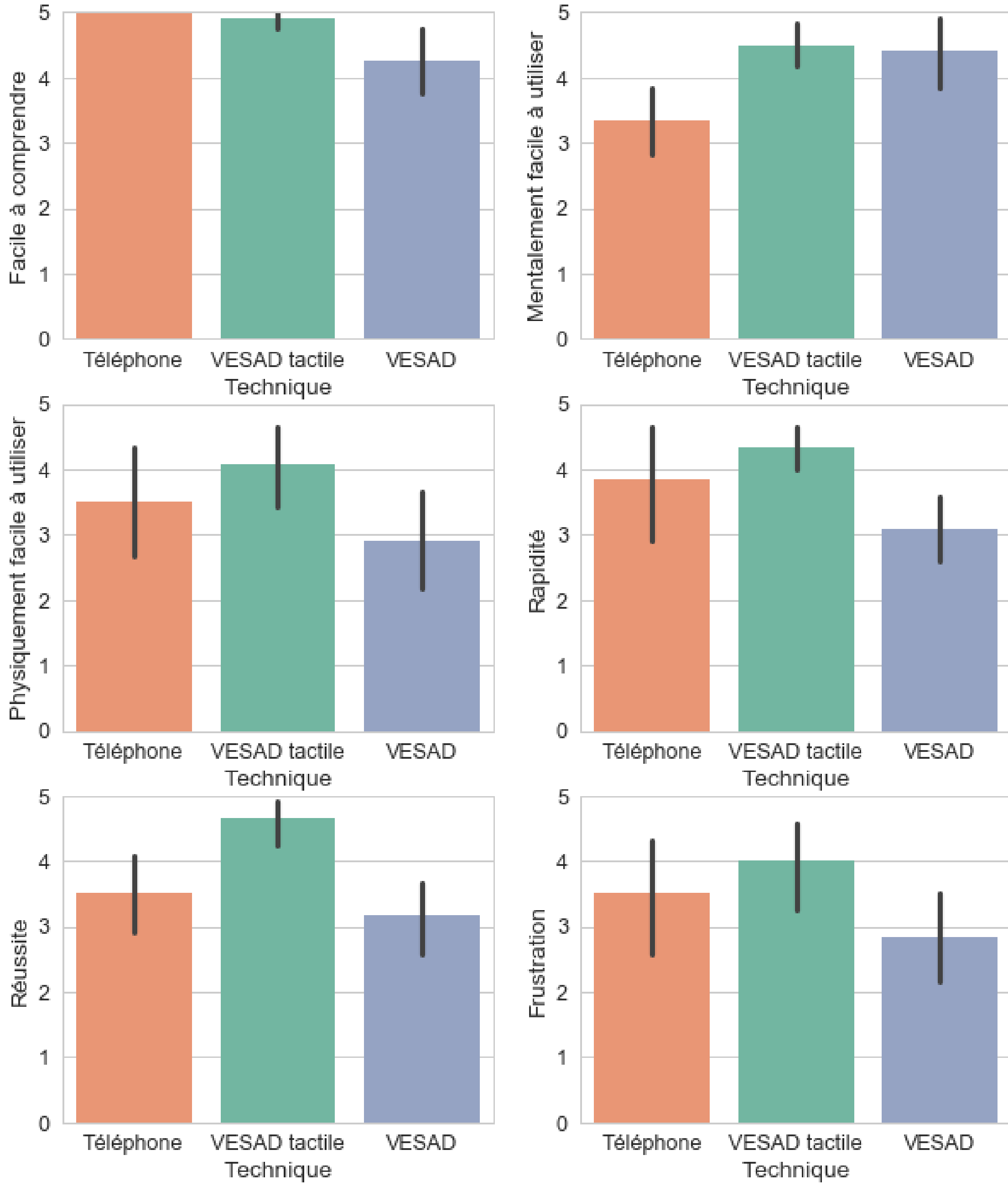
Display the mean of the ranks with the standard deviation for each question among TECHNIQUE:

```
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', :]]
```

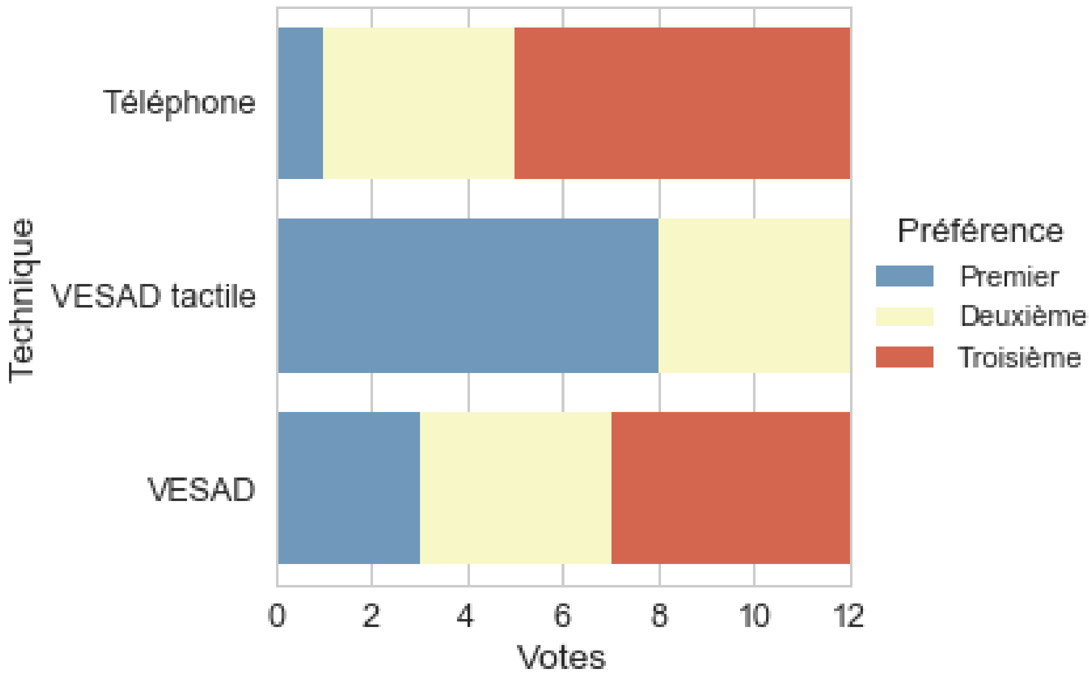
```
Out[30]:               Facile à comprendre Mentalement facile à utiliser  \
        Technique
        Téléphone                5.00±0.00                     3.33±0.85
        VESAD tactile            4.92±0.28                     4.50±0.65
        VESAD                    4.25±0.83                     4.42±0.95

                      Physiquement facile à utiliser   Rapidité   Réussite  \
        Technique
        Téléphone                          3.50±1.50  3.83±1.52  3.50±1.04
        VESAD tactile                      4.08±1.19  4.33±0.62  4.67±0.62
        VESAD                              2.92±1.32  3.08±0.95  3.17±1.07
```

```
                 Frustration Préférence
Technique
Téléphone          3.50±1.61  2.50±0.65
VESAD tactile      4.00±1.15  1.33±0.47
VESAD              2.83±1.21  2.17±0.80
```

In [31]:
```python
(fig, axs) = plot_ranks('technique', ranks_dvs.columns[0:-1])
fig.savefig('ranks.png', dpi=300, bbox_inches='tight')
```



```
In [32]: (fig, axs) = plot_ranks_distributions('technique', ['preference'])

         ax_handles, ax_labels = axs[0].get_legend_handles_labels()
         legend = axs[0].legend(ax_handles[::-1], labels['preferences'], loc='center left',\
                                bbox_to_anchor=(1, 0.5), title=ranks_dvs['preference']['label'])
         fix_legend_fontsize(legend)

         fig.savefig('preferences.png', dpi=300, bbox_inches='tight')
```



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=[labels['iv'], labels['iv_value'] + ' (1)',\
                                                  labels['iv_value'] + ' (2)', labels['dv'],\
                                                  labels['mean_difference'], labels['mean_differenc
                                                  labels['t_statistic'], labels['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=[labels['iv'], labels['dv'],\
                                                       'Kruskal-Wallis H', labels['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=[labels['iv'], labels['iv_value'] + ' (1)',\
                                              labels['iv_value'] + ' (2)', labels['dv'],\
                                              'Mann-Whitney U', labels['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))
```

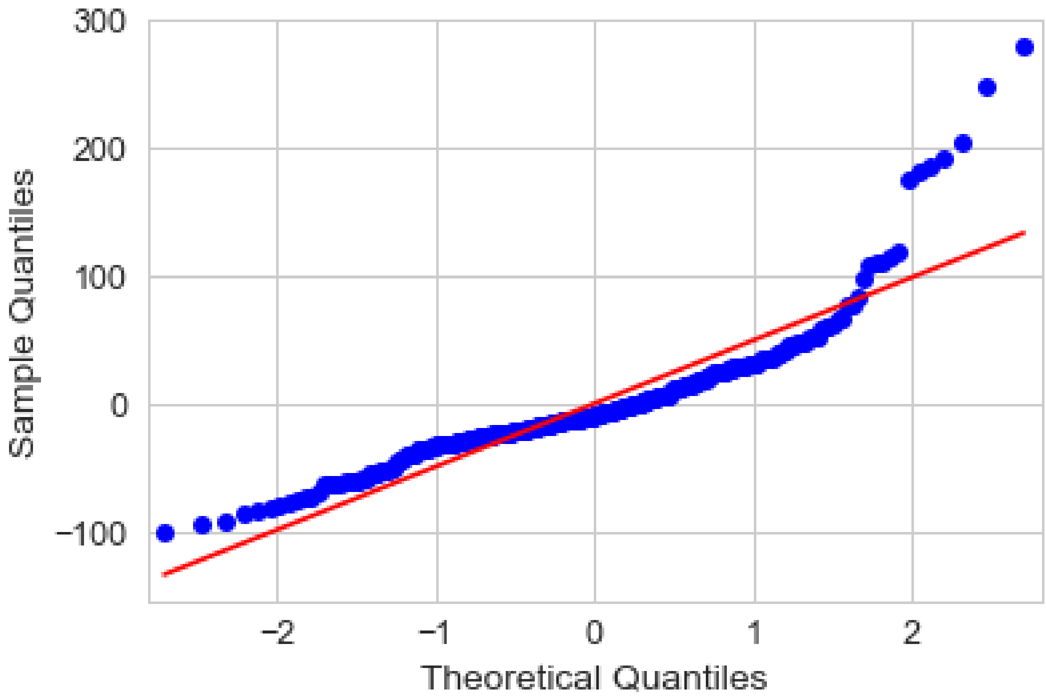
```
        return (fig, axs)
```

3.1 3.1. Task completion time

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 [40]: 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)

         fig = sm.qqplot(tct_model.resid, line='s')
         fig.savefig('tct_anova.png', dpi=300, bbox_inches='tight')
         plt.show()

         tct_model_anova
```



Out[40]:

	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 [41]:
```
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[41]:
```
   Variable indépendante Valeur variable indépendante (1)  \
0              Technique                        Téléphone
1              Technique                        Téléphone
2              Technique                    VESAD tactile
3              Technique                        Téléphone
4              Technique                        Téléphone
5              Technique                    VESAD tactile
6              Technique                        Téléphone
7              Technique                        Téléphone
8              Technique                    VESAD tactile
9              Technique                        Téléphone
10             Technique                        Téléphone
11             Technique                    VESAD tactile
12             Technique                        Téléphone
13             Technique                        Téléphone
14             Technique                    VESAD tactile

   Valeur variable indépendante (2)      Variable dépendante  \
0                     VESAD tactile  Temps de complétion (s)
1                             VESAD  Temps de complétion (s)
2                             VESAD  Temps de complétion (s)
3                     VESAD tactile  Temps de complétion (s)
4                             VESAD  Temps de complétion (s)
5                             VESAD  Temps de complétion (s)
6                     VESAD tactile  Temps de complétion (s)
7                             VESAD  Temps de complétion (s)
8                             VESAD  Temps de complétion (s)
9                     VESAD tactile  Temps de complétion (s)
10                            VESAD  Temps de complétion (s)
```

```
11                          VESAD  Temps de complétion (s)
12                  VESAD tactile  Temps de complétion (s)
13                          VESAD  Temps de complétion (s)
14                          VESAD  Temps de complétion (s)

    Différence des moyennes  Différence des moyennes (%)  Statistique T  \
0                 21.840362                    32.666819       4.600061
1                -49.467731                   -35.803106      -5.669051
2                -71.308093                   -51.610437      -8.608496
3                 24.168476                    36.724278       2.549015
4                -77.353492                   -46.227392      -3.835002
5               -101.521968                   -60.670769      -5.199072
6                 26.892856                    41.152781       2.379929
7                -43.047900                   -31.819082      -2.450544
8                -69.940756                   -51.697078      -4.614999
9                  9.832919                    13.926232       1.170723
10               -26.990175                   -25.123437      -2.028633
11               -36.823094                   -34.276275      -2.721200
12                26.467198                    40.306314       2.970482
13               -50.479357                   -35.396365      -3.005064
14               -76.946555                   -53.955290      -4.776990

        Valeur p        Condition
0   2.887821e-05
1   3.951549e-07
2   4.178800e-14
3   1.936788e-02  (Grand, Proche)
4   8.149009e-04  (Grand, Proche)
5   2.246879e-05  (Grand, Proche)
6   2.482784e-02    (Grand, Loin)
7   2.265802e-02    (Grand, Loin)
8   7.898347e-05    (Grand, Loin)
9   2.477394e-01  (Petit, Proche)
10  5.175809e-02  (Petit, Proche)
11  1.372939e-02  (Petit, Proche)
12  7.855825e-03    (Petit, Loin)
13  7.855825e-03    (Petit, Loin)
14  5.551087e-05    (Petit, Loin)
```

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:

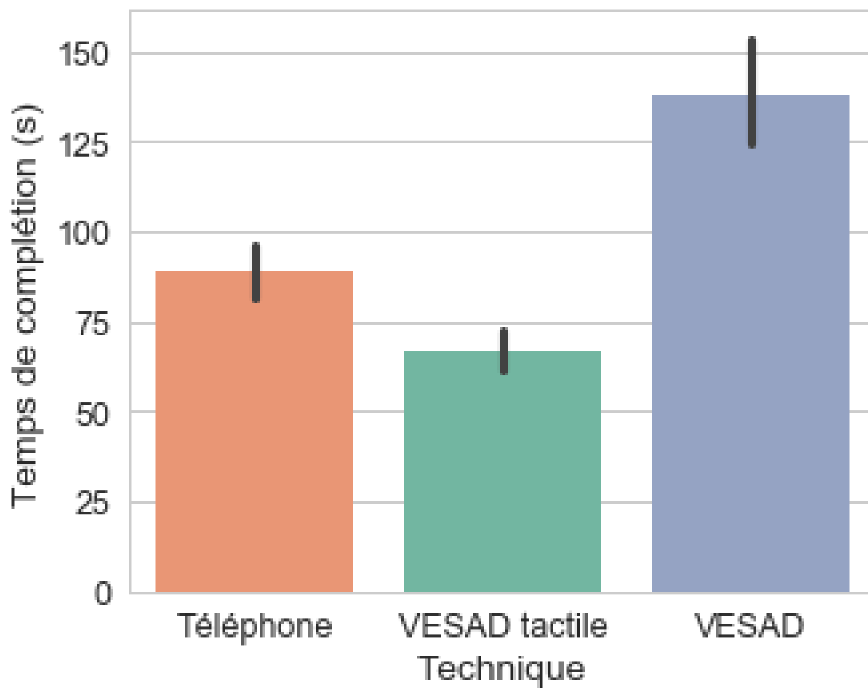
- For *Big* text size and *Near* distance:
  - *PhoneInArOut* is 24 s faster than *PhoneOnly* ($p < 0.019$, 37% faster);
  - *PhoneInArOut* is 102 s faster than *MidAirInArOut* ($p < 0.0001$, 61% faster);
  - *PhoneOnly* is 77 s faster than *MidAirInArOut* ($p = 0.0008$, 46% faster).
- For *Big* text size and *Far* distance:
  - *PhoneInArOut* is 27 s faster than *PhoneOnly* ($p < 0.022$, 41% faster);

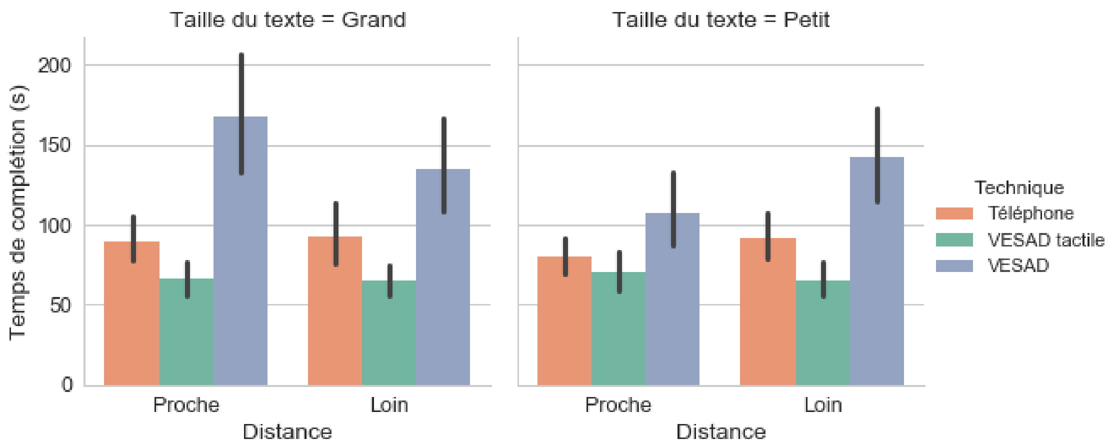
- *PhoneInArOut* is 70 s faster than *MidAirInArOut* ($p < 0.0001$, 52% faster);
- *PhoneOnly* is 43 s faster than *MidAirInArOut* ($p = 0.023$, 32% faster).

- 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 [42]:
```
(fig, axs) = plot_trials('total_time', [['technique']])
fig.savefig('tct.png', dpi=300, bbox_inches='tight')

g = sns.factorplot(x=distance['label'], y=trials_dvs['total_time'], col=text_size['label'], hue
                   data=trials, palette=technique['palette'], kind='bar')
g.savefig('tct_2.png', dpi=300, bbox_inches='tight')
```

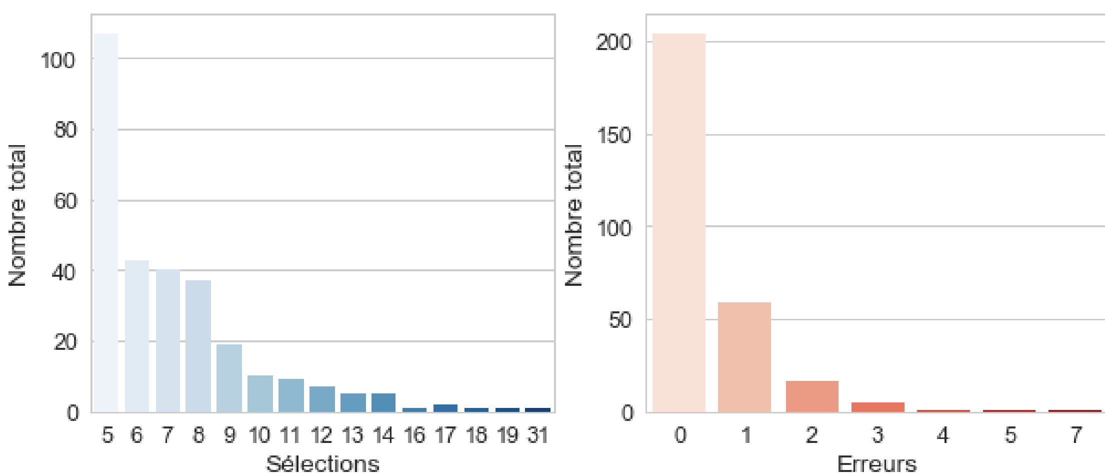




3.2 3.2. Error rate

Visualize the SELECTIONS and ERRORS distributions:

In [43]:
```
(fig, axs) = subplots(2)

sns.countplot(x=trials_dvs['selections_count'], data=trials, palette='Blues', ax=axs[0])
axs[0].set(ylabel=labels['count'])

sns.countplot(x=trials_dvs['errors'], data=trials, palette='Reds', ax=axs[1])
axs[1].set(ylabel=labels['count'])

fig.savefig('selections_errors_distributions.png', dpi=300, bbox_inches='tight')
```



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

```
Out[44]:   Variable indépendante Variable dépendante  Kruskal-Wallis H  Valeur p
         0             Technique          Sélections         20.015292  0.000217
         1       Taille du texte          Sélections          0.000810  0.977297
         2              Distance          Sélections          0.329777  0.754387
         3                Groupe          Sélections         19.648369  0.000217
         4             Technique             Erreurs          6.510257  0.077152
         5       Taille du texte             Erreurs          0.063506  0.915472
         6              Distance             Erreurs          0.437007  0.754387
         7                Groupe             Erreurs         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):

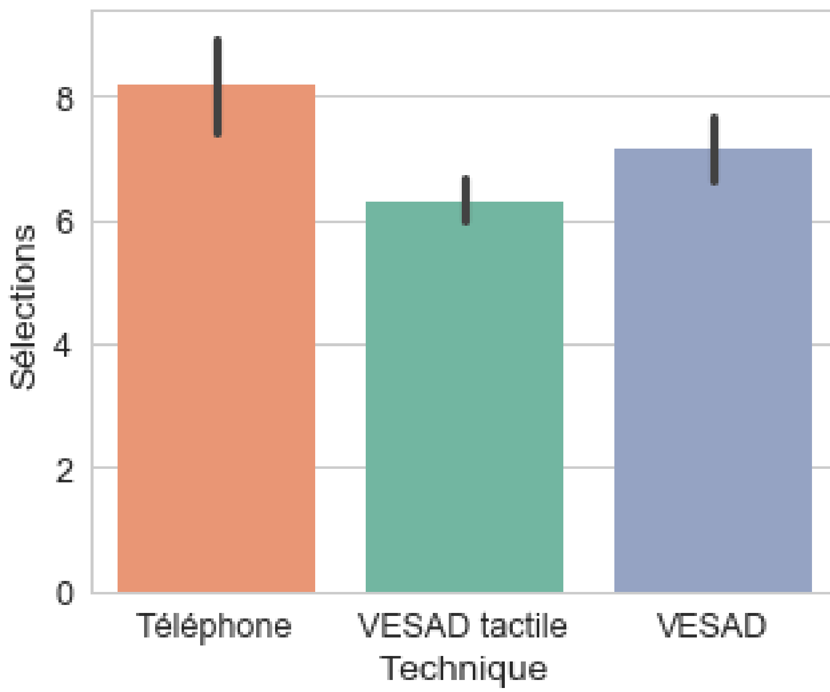
```
In [45]: test_pairwise_non_normal_trials(['selections_count', 'errors'], ['technique'])
```

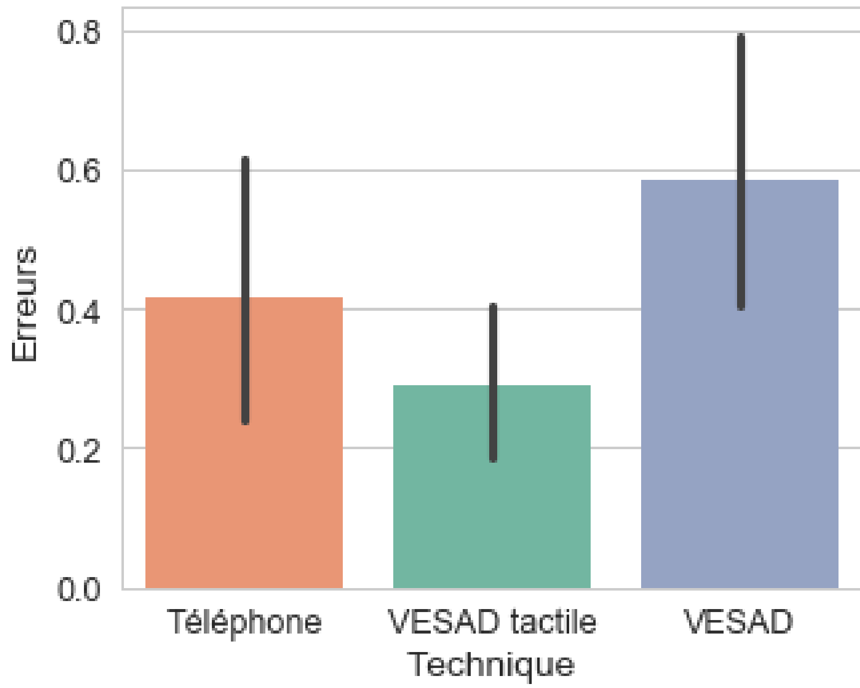
```
Out[45]:   Variable indépendante Valeur variable indépendante (1)  \
         0             Technique                        Téléphone
         1             Technique                        Téléphone
         2             Technique                    VESAD tactile
         3             Technique                        Téléphone
         4             Technique                        Téléphone
         5             Technique                    VESAD tactile

           Valeur variable indépendante (2) Variable dépendante  Mann-Whitney U  \
         0                    VESAD tactile          Sélections          2923.0
         1                            VESAD          Sélections          3733.5
         2                            VESAD          Sélections          3850.5
         3                    VESAD tactile             Erreurs          4502.5
         4                            VESAD             Erreurs          4006.5
         5                            VESAD             Erreurs          3869.0

            Valeur p
         0  0.000020
         1  0.020678
         2  0.028448
         3  0.358201
         4  0.034230
         5  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.

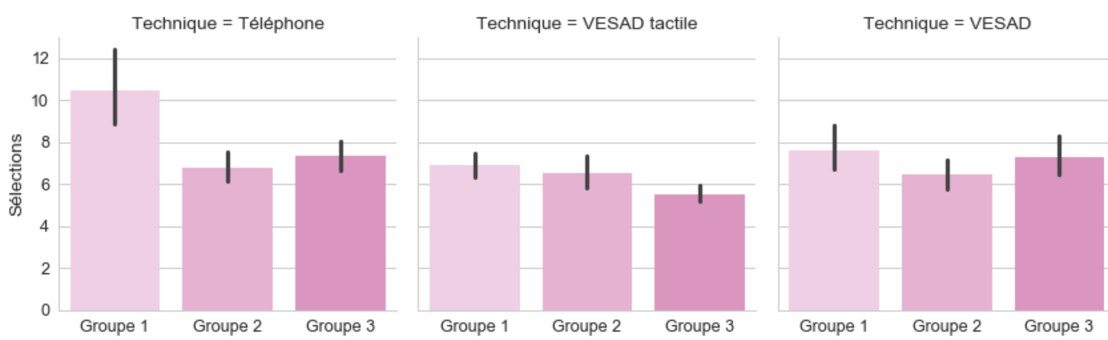
- Participants made significant more errors with *MidAirInArOut* rather than *PhoneOnly* (p=0.034) or *PhoneInArOut* (p=0.021), but these latter two do not differ significantly. With *MidAirInArOut*, users sometimes dropped items in the wrong container, not voluntarily but because they were too zoomed out and/or the sensing limitations of the Leap Motion made it difficult to successfully aim at targets, especially when arms were crossed.

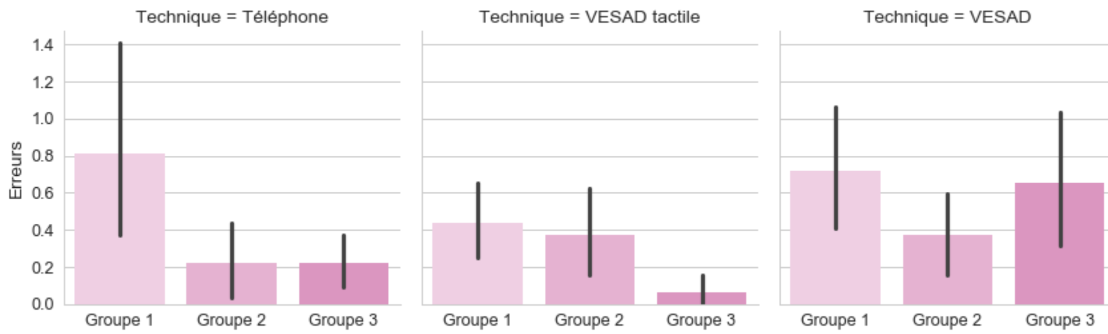
In [46]:
```
(fig, axs) = plot_trials('selections_count', [['technique']])
fig.savefig('selections.png', dpi=300, bbox_inches='tight')

(fig, axs) = plot_trials('errors', [['technique']])
fig.savefig('errors.png', dpi=300, bbox_inches='tight')
```






```
In [47]: g = sns.factorplot(x=ordering['label'], y=trials_dvs['selections_count'], col=technique['label
                            palette=ordering['palette'], kind='bar', data=trials)
         g.set_axis_labels('')
         g.savefig('selections_ordering.png', dpi=300, bbox_inches='tight')


         g = sns.factorplot(x=ordering['label'], y=trials_dvs['errors'], col=technique['label'],\
                            palette=ordering['palette'], kind='bar', data=trials)
         g.set_axis_labels('')
         g.savefig('errors_ordering.png', dpi=300, bbox_inches='tight')
```

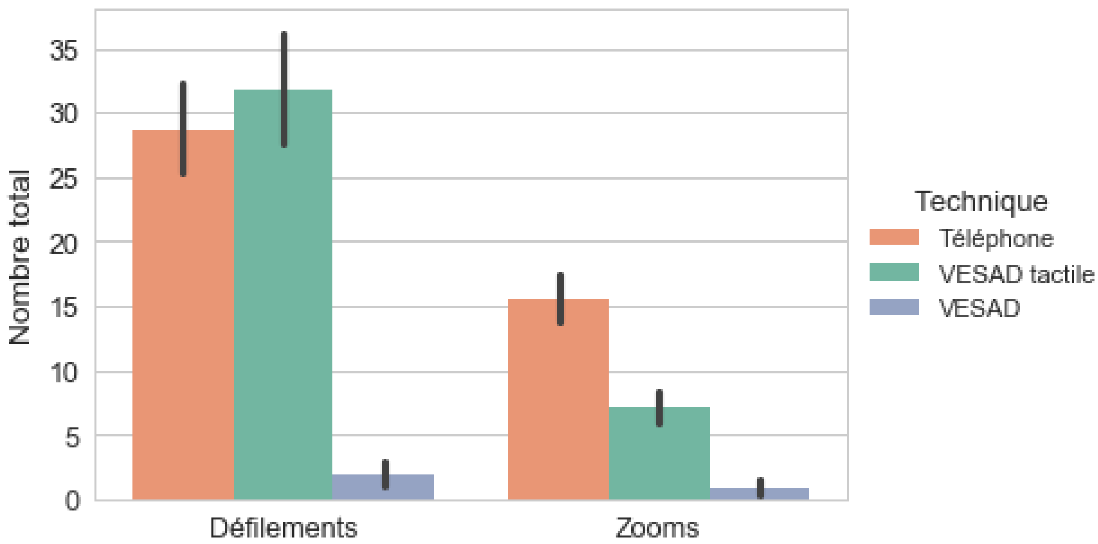




PhoneOnly seems to be more sensitive in terms of ERRORS to being the first technique tested by users.

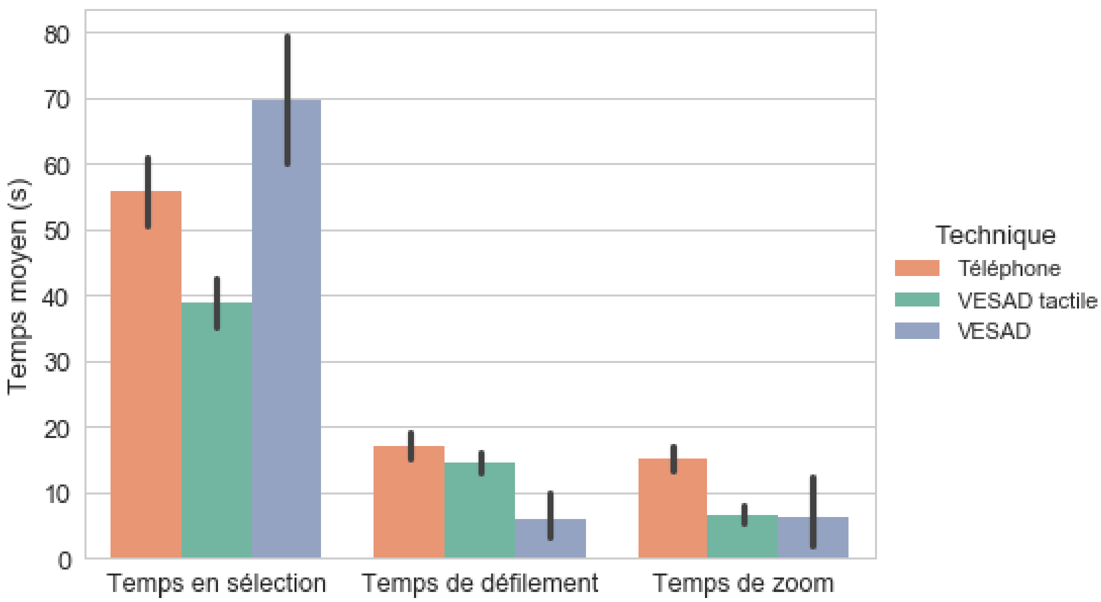
3.3 3.3. Navigation

In [48]:
```
trial_counts = melt_trials(var_name=labels['category'], value_name=labels['count'],\
                           value_vars=[trials_dvs['pan_count'], trials_dvs['zoom_count']])


fig = plt.figure(figsize=(6,4))

ax = sns.barplot(x=labels['category'], y=labels['count'], hue=technique['label'],\
                 palette=technique['palette'], data=trial_counts)
legend = ax.legend(loc='center left', bbox_to_anchor=(1, 0.5), title=technique['label'])
fix_legend_fontsize(legend)
ax.set(xlabel='')

plt.show()
fig.savefig('navigation_count.png', dpi=300, bbox_inches='tight')


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[48]:                Nombre total
         Catégorie       Défilements        Zooms
         Technique
         Téléphone       28.70±18.56  15.53±8.63
         VESAD tactile   31.81±20.82   7.14±6.19
         VESAD            1.83±4.79   0.83±3.20
```

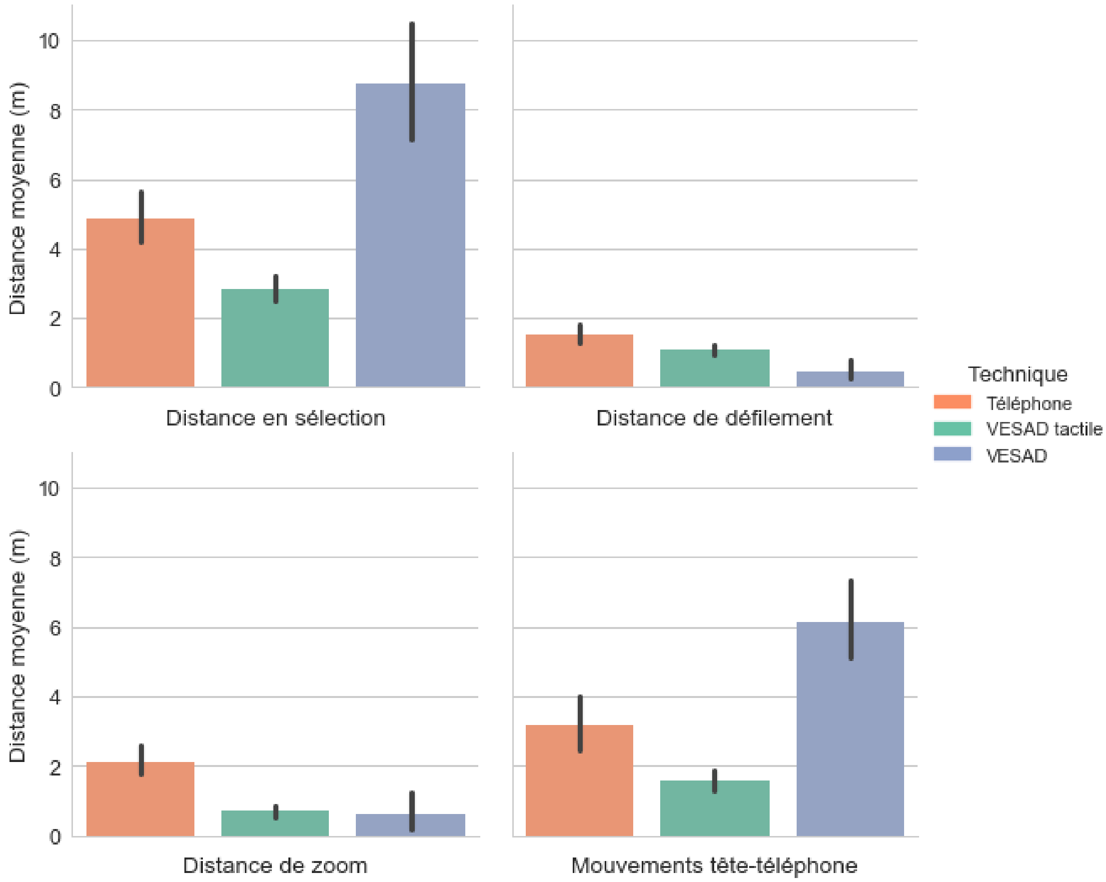
```
In [49]: trial_times = melt_trials(var_name=labels['category'], value_name=labels['time'],\
                                   value_vars=[trials_dvs['selections_time'], trials_dvs['pan_time'], t


         fig = plt.figure(figsize=(7,5))

         ax = sns.barplot(x=labels['category'], y=labels['time'], hue=technique['label'], palette=techn
                          data=trial_times)
         legend = ax.legend(loc='center left', bbox_to_anchor=(1, 0.5), title=technique['label'])
         fix_legend_fontsize(legend)
         ax.set(xlabel='')

         plt.show()
         fig.savefig('navigation_time.png', dpi=300, bbox_inches='tight')


         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[49]:                  Temps moyen (s)
         Catégorie     Temps en sélection Temps de défilement Temps de zoom
         Technique
         Téléphone            55.74±27.54         17.12±10.66    15.18±9.00
         VESAD tactile        38.98±19.21          14.53±8.10     6.70±6.83
         VESAD                69.68±47.17          5.97±16.56    6.24±26.31
```

```
In [50]: 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']])

         g = sns.factorplot(x=technique['label'], y=labels['distance'], col=labels['category'], data=tri
                            palette=technique['palette'], kind='bar', col_wrap=2)

         g.set_titles('{col_name}') # Replace subplot titles
         for ax in g.axes:
             ax.title.set_position([0.5, -0.12])

         g.set_axis_labels('') # Custom legend
         g.set_xticklabels([])
         legend_handles = [patches.Patch(color=color, label=value)\
                           for value, color in zip(technique['categorical'], technique['palette'])]
         legend = plt.legend(handles=legend_handles, loc='center left', bbox_to_anchor=(1, 1.1), title=
         fix_legend_fontsize(legend)

         plt.show()
         g.savefig('navigation_distance.png', dpi=300, bbox_inches='tight')

         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[50]:                Distance moyenne (m)                                          \
         Catégorie     Distance en sélection Distance de défilement Distance de zoom
         Technique
         Téléphone                 4.87±3.74              1.50±1.31        2.11±1.99
         VESAD tactile             2.83±1.89              1.09±0.76        0.69±0.74
         VESAD                     8.72±7.89              0.47±1.30        0.62±2.77


         Catégorie     Mouvements tête-téléphone
         Technique
         Téléphone                     3.18±3.55
         VESAD tactile                 1.57±1.37
         VESAD                         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.