ANNEXE II

ANALYSE DÉTAILLÉE DE L'EXPÉRIENCE

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

0.1 1. Data preparation

Configuration :

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

        import seaborn as sns
        import matplotlib as mpl
        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

        from IPython.html.services.config import ConfigManager
```

```
C:\Users\Erwan\Miniconda\envs\master-thesis\lib\site-packages\IPython\html.py:14: ShimWarning: The
  "`IPython.html.widgets` has moved to `ipywidgets`.", ShimWarning)
```

```
In [2]: # Notebook configuration
        %matplotlib inline

        # Ruler
```

```
ip = get_ipython()
cm = ConfigManager(parent=ip)
cm.update('notebook', {"ruler_column": [80]})

# Style and size of the figures
subplotsize = (5, 4)
legend_fontsize = sns.plotting_context('notebook')['axes.labelsize']
legend_title_fontsize = legend_fontsize + 1
sns.set(context='notebook', style='whitegrid', font_scale=1.25,
        rc={'legend.fontsize': legend_fontsize, 'savefig.dpi': 300,
            'savefig.bbox': 'tight'})

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

Data are loaded in the following variables: - The ranks of the participants from the post-questionary: `ranks` - Trials of the participants: `raw_trials` - Trials averaged so that each participants ends up with a single observation (Tip 9 of Dragicevic, 2016) : `trials` > Dragicevic, P. (2016). Fair statistical communication in HCI. In Modern Statistical Methods for HCI (pp. 291-330). Springer, Cham.

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

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

```
In [4]: # IVs
        if language == 'Français':
            iv_labels = ['IHM', 'Taille du texte', 'Distance', 'Groupe']
        else:
            iv_labels = ['Technique', 'Text Size', 'Distance', 'Ordering']

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

        # Shortcuts
        technique = ivs['technique']
        text_size = ivs['text_size']
        distance = ivs['distance']
        ordering = ivs['ordering']

        # Setup categories and palettes
        palette = sns.color_palette('Set2', 8)
        for iv_id, iv_label in zip(ivs, iv_labels):
            iv_categories = trials.drop_duplicates(iv_id)\
                                  .sort_values([iv_id + '_id'])[iv_id]
            iv_categorical = pd.Categorical(iv_categories, iv_categories, ordered=True)
            ivs[iv_id] = [iv_label, iv_categorical, palette]
```

```
technique['palette'] = [palette[1], palette[0], palette[2]]
text_size['palette'] = sns.light_palette(palette[6], 3)[1:3]
distance['palette'] = sns.light_palette(palette[4], 3)[1:3]
ordering['palette'] = [palette[1], palette[0], palette[2]]
```

```
In [5]: # 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)

        # Shortcuts
        participant_id = 'Participant'
        total_time = trials_dvs['total_time']
```

```
In [6]: # 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(ranks_dv_labels)

        RdYlBu = sns.color_palette('RdYlBu', 5)
        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)), ordered=True)
ranks_dvs.at['palette', 'preference'] = [RdYlBu[4], RdYlBu[2], RdYlBu[0]]
```

Clean the data:

```
In [7]: # Setup columns of trials and ranks
        columns = []
        for column in trials.columns:
            if (column in ivs.columns):
                columns.append(ivs[column]['label'])
            elif (column in trials_dvs.index):
                columns.append(trials_dvs[column])
            else:
                columns.append(column)
        columns[0] = participant_id

        trials.columns = columns
        ranks.columns = [participant_id, ordering['label'], technique['label']]\
                        + ranks_dvs.loc['label', :].tolist()

        # Setup translated column values of trials and ranks
        for iv_id in ivs:
            iv = ivs[iv_id]
            trials[iv['label']] = trials[iv['label']].astype(iv['categorical'])
            raw_trials[iv_id] = raw_trials[iv_id].astype(iv['categorical'])

        for iv_id in ['technique', 'ordering']:
            iv = ivs[iv_id]
            ranks[iv['label']] = ranks[iv['label']].astype(iv['categorical'])

        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']
        else:
            ordering['categorical'].categories = ['Group 1', 'Group 2', 'Group 3']

        for iv_id in ivs:
            iv = ivs[iv_id]
            trials[iv['label']].cat.categories = iv['categorical'].categories

        for iv_id in ['technique', 'ordering']:
            iv = ivs[iv_id]
            ranks[iv['label']].cat.categories = iv['categorical'].categories
```

```
In [8]: # Some participants are non valid or don't have complete measures
        invapars = [0, 4]
        ranks = ranks[~ranks[participant_id].isin(invapars)].reset_index(drop=True)
```

```
trials = trials[~trials[participant_id].isin(invapars)].reset_index(drop=True)
raw_trials = raw_trials[~raw_trials['participant_id'].isin(invapars)
                        ].reset_index(drop=True)
```

In [9]:
```
# Some participants have wrong head phone mesures
for dist in ['absolute_head_phone_distance', 'signed_head_phone_distance']:
    trials.loc[trials[trials_dvs[dist]] == 0, trials_dvs[dist]] = np.nan
    raw_trials.loc[raw_trials[dist] == 0, dist] = np.nan
```

In [10]:
```
# 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)
```

In [11]:
```
# Average trials to have one observation per participant
trials = trials.groupby([participant_id] + ivs.loc['label', :].tolist(),
                        observed=True).mean().reset_index()
raw_trials = raw_trials.groupby(['participant_id'] + ivs.columns.tolist(),
                        observed=True).mean().reset_index()
```

Utilities:

In [12]:
```
# Figure labels in the selected language
labels = pd.Series()

if language == 'Français':
    labels['category'] = 'Catégorie'
    labels['count'] = 'Nombre total'
    labels['distance'] = 'Distance moyenne (m)'
    labels['dv'] = 'VD'
    labels['iv'] = 'VI'
    labels['iv_value'] = 'Valeur VI'
    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'] = 'Participants'
else:
    labels['category'] = 'Category'
    labels['count'] = 'Count'
    labels['distance'] = 'Mean Distance (m)'
    labels['dv'] = 'DV'
    labels['iv'] = 'IV'
    labels['iv_value'] = 'IV 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'] = 'Participants'
```

In [13]:
```
def mean_ci(x, which=95, n_boot=1000):
    """Returns the confidence interval of the mean"""
    x2 = [i for i in x if not np.isnan(i)]
    boots = sns.algorithms.bootstrap(x2, n_boot=n_boot)
    return sns.utils.ci(boots, which=which)
```

In [14]:
```
def print_mean_ci(x, ci_which=95):
    """Returns a string containing the mean with the CI of x"""
    ci1, ci2 = mean_ci(x, which=ci_which)
    return '{:.2f} [{:.2f}, {:.2f}]'.format(np.mean(x), ci1, ci2)
```

In [15]:
```
def exp_mean(x):
    return np.exp(np.mean(x))
```

In [16]:
```
def print_exp_mean_ci(x, ci_which=95):
    ci1, ci2 = np.exp(mean_ci(x, which=ci_which))
    return '{:.2f} [{:.2f}, {:.2f}]'.format(exp_mean(x), ci1, ci2)
```

In [17]:
```
def geometric_mean(x):
    return exp_mean(np.log(x))
```

In [18]:
```
def print_geo_mean_ci(x, ci_which=95):
    ci1, ci2 = np.exp(mean_ci(np.log(x), which=ci_which))
    return '{:.2f} [{:.2f}, {:.2f}]'.format(geometric_mean(x), ci1, ci2)
```

In [19]:
```
def mean_difference(a, b):
    """Returns the mean difference value and percentage between a and b"""
    mean_diff = np.mean(a) - np.mean(b)
    mean_diff_percentage = mean_diff / np.mean(b) * 100
    return (mean_diff, mean_diff_percentage)
```

In [20]:
```
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(), alpha=alpha,
                          method=correction_method)
        data[labels['p_value']] = p_values_corrected
```

In [21]:
```
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, subplotsize[1]*nrows),
                            *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 [22]: def fix_legend_fontsize(ax_legend, fontsize=legend_title_fontsize):
             plt.setp(ax_legend.get_title(), fontsize=fontsize)
```

```
In [23]: def config_legend(ax, iv_id, fontsize=legend_title_fontsize):
             legend = ax.legend(title=ivs[iv_id]['label'], frameon=True,
                                loc='center left', bbox_to_anchor=(1, 0.5))
             fix_legend_fontsize(legend)
             return legend
```

0.2 2. Participant ranks

Some functions for the analysis:

```
In [24]: def get_ranks_count(iv_index, dv_index):
             iv, dv = ivs[iv_index], ranks_dvs[dv_index]

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

             # Merge and remove duplicated defaults
             ranks_counts = pd.concat([ranks_counts, default_counts])
             ranks_counts = ranks_counts[~ranks_counts.index.duplicated(keep='first')]

             ranks_counts.sort_index(inplace=True)
             ranks_counts.index = ranks_counts_index # Restore the index
             return ranks_counts
```

```
In [25]: def cumulated_barplot(data, palette, **args):
             for row_id, row in data.iloc[::-1].iterrows():
                 sns.barplot(y=data.columns, x=row, label=row_id,
                             color=palette[row_id-1], orient='h', **args)
```

```
In [26]: def plot_ranks_distributions(iv_id, dv_ids):
             iv = ivs[iv_id]
```

```
    fig, axs = subplots(len(dv_ids))
    for dv_id, ax in zip(dv_ids, axs):
        dv = ranks_dvs[dv_id]

        cumulated_ranks_count = get_ranks_count(iv_id, dv_id).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['label'],
                           fontsize=legend_fontsize-2)
        fix_legend_fontsize(legend)

    fig.tight_layout(h_pad=1) # Add padding to avoir legend and labels overlap
    return (fig, axs)
```

```
In [27]: def plot_ranks(iv_id, dv_ids, estimator=np.mean):
             iv = ivs[iv_id]

             fig, axs = subplots(len(dv_ids))
             for dv_id, ax in zip(dv_ids, axs):
                 dv = ranks_dvs[dv_id]

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

             return (fig, axs)
```

```
In [28]: def rank_samples(iv_id, dv_id):
             """Returns the list of ranks (DV) for each IV value"""
             samples = []
             iv = ivs[iv_id]
             for iv_value in iv['categorical']:
                 dv_label = ranks_dvs[dv_id]['label']
                 sample = ranks[ranks[iv['label']] == iv_value][dv_label]\
                          .reset_index(drop=True)
                 samples.append(sample)
             return samples
```

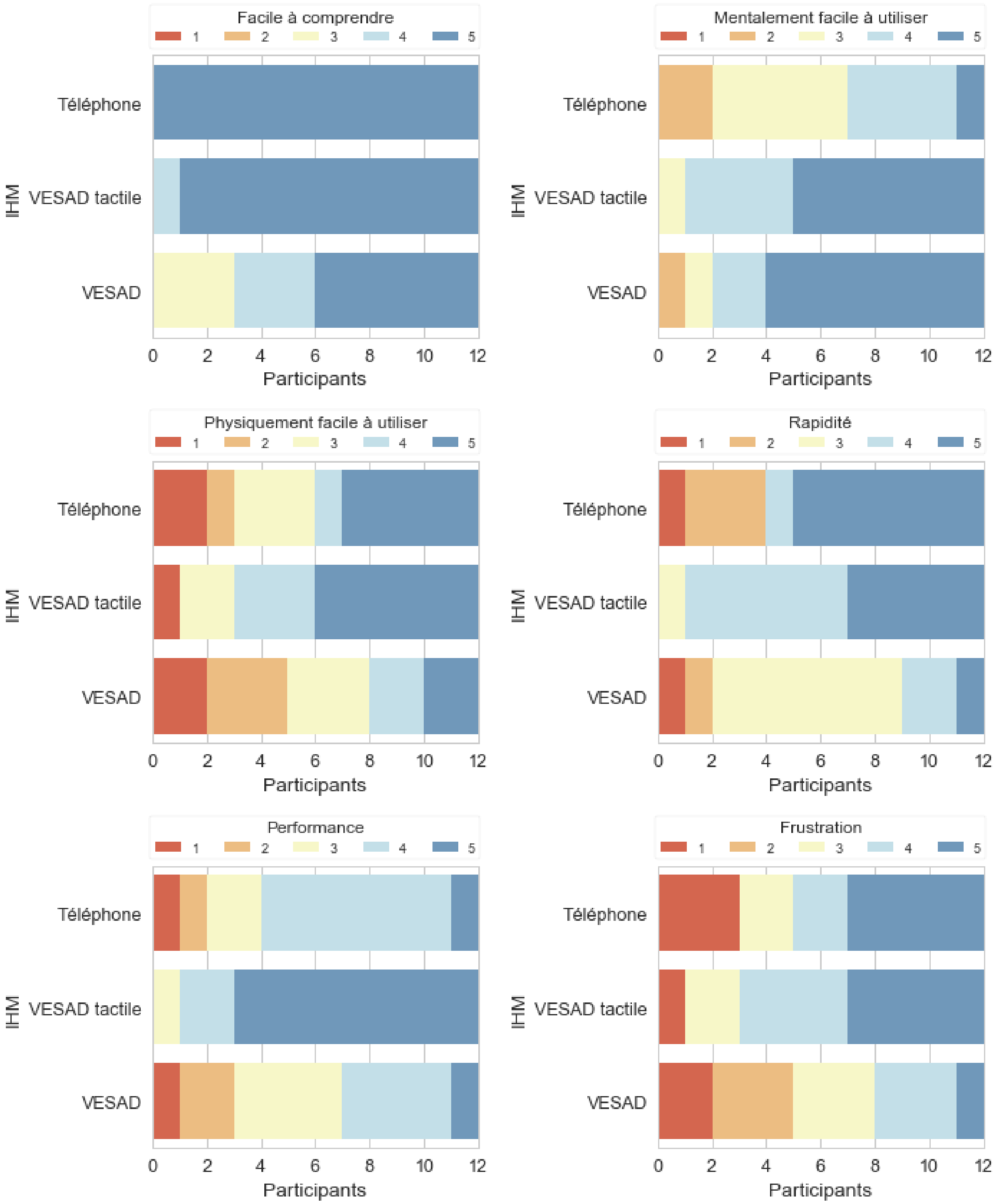
```
In [29]: def test_ranks(iv_id, dv_ids, **args):
             results = []
             iv = ivs[iv_id]

             for dv_id in dv_ids:
```

```
            dv = ranks_dvs[dv_id]
            samples = rank_samples(iv_id, dv_id)
            H, p = stats.kruskal(*samples)
            results.append([iv['label'], dv['label'], H, p])

        results = pd.DataFrame(results, columns=[labels['iv'], labels['dv'],
                                                 'Kruskal-Wallis H',
                                                 labels['p_value']])
        p_values_correction(results, **args)
        return results
```

```
In [30]: def test_pairwise_ranks(iv_id, dv_ids, **args):
             iv = ivs[iv_id]
             iv_category_ids = range(len(iv['categorical']))

             results = []
             for dv_id in dv_ids:
                 dv = ranks_dvs[dv_id]
                 samples = rank_samples(iv_id, dv_id)
                 sample_pairs = itertools.combinations(iv_category_ids, 2)
                 for id1, id2 in sample_pairs:
                     U, p = stats.mannwhitneyu(samples[id1], samples[id2])
                     mean_diff, mean_diff_per = mean_difference(samples[id1], samples[id2])
                     results.append([iv['label'], iv['categorical'][id1],
                                     iv['categorical'][id2], dv['label'],
                                     mean_diff, mean_diff_per, U, p])

             columns = [labels['iv'], labels['iv_value'] + ' 1',
                         labels['iv_value'] + ' 2', labels['dv'],
                         labels['mean_difference'], labels['mean_difference_percentage'],
                         'Mann-Whitney U', labels['p_value']]
             results = pd.DataFrame(results, columns=columns)
             p_values_correction(results, **args)
             return results
```

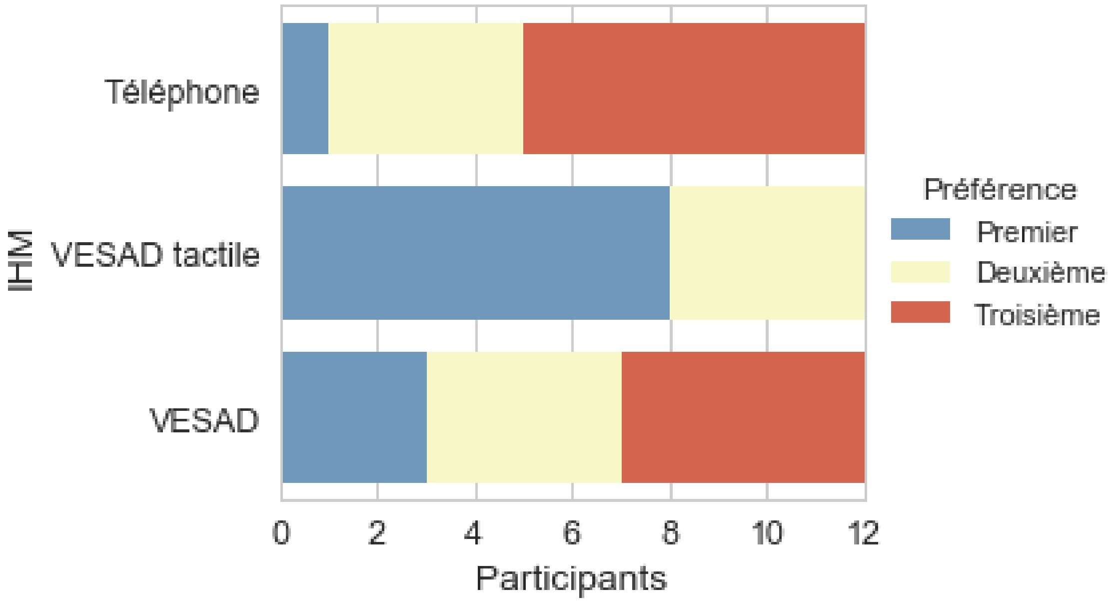
We display the rank and preference distributions:

```
In [31]: (fig, axs) = plot_ranks_distributions('technique', ranks_dvs.columns[0:-1])
         fig.savefig('ranks_distributions.png')
```




```
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_distribution.png')
```



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

```
In [33]: test_ranks('technique', ranks_dvs, correction_method='fdr_bh')
```

```
Out[33]:     VI                              VD  Kruskal-Wallis H  Valeur p
         0  IHM             Facile à comprendre         11.001420  0.007497
         1  IHM   Mentalement facile à utiliser         10.905742  0.007497
         2  IHM  Physiquement facile à utiliser          4.148121  0.125674
         3  IHM                        Rapidité          7.130846  0.039599
         4  IHM                     Performance         13.784269  0.006303
         5  IHM                     Frustration          4.500057  0.122962
         6  IHM                      Préférence         12.638889  0.006303
```

All the questions, except Physically Easy to Use and Frustration, are statistically significants: Easy to Understand ($p = 0.007$), Mentally Easy to Use ($p = 0.007$), Subjective Speed ($p = 0.04$), Subjective Performance ($p = 0.006$), Preference ($p = 0.006$).

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

```
In [34]: test_pairwise_ranks('technique', ['easy_understand', 'mentally_easy_use',
                                          'could_go_fast', 'subjective_performance',
                                          'preference'], correction_method='fdr_bh')
```

```
Out[34]:     VI   Valeur VI 1   Valeur VI 2                        VD  \
         0  IHM     Téléphone  VESAD tactile       Facile à comprendre
         1  IHM     Téléphone          VESAD       Facile à comprendre
```

```
2   IHM  VESAD tactile          VESAD            Facile à comprendre
3   IHM      Téléphone  VESAD tactile  Mentalement facile à utiliser
4   IHM      Téléphone          VESAD  Mentalement facile à utiliser
5   IHM  VESAD tactile          VESAD  Mentalement facile à utiliser
6   IHM      Téléphone  VESAD tactile                       Rapidité
7   IHM      Téléphone          VESAD                       Rapidité
8   IHM  VESAD tactile          VESAD                       Rapidité
9   IHM      Téléphone  VESAD tactile                    Performance
10  IHM      Téléphone          VESAD                    Performance
11  IHM  VESAD tactile          VESAD                    Performance
12  IHM      Téléphone  VESAD tactile                     Préférence
13  IHM      Téléphone          VESAD                     Préférence
14  IHM  VESAD tactile          VESAD                     Préférence

    Différence des moyennes  Différence des moyennes (%)  Mann-Whitney U  \
0                  0.083333                     1.694915            66.0
1                  0.750000                    17.647059            36.0
2                  0.666667                    15.686275            40.5
3                 -1.166667                   -25.925926            23.0
4                 -1.083333                   -24.528302            28.5
5                  0.083333                     1.886792            69.5
6                 -0.500000                   -11.538462            70.5
7                  0.750000                    24.324324            48.5
8                  1.250000                    40.540541            21.0
9                 -1.166667                   -25.000000            22.5
10                 0.333333                    10.526316            57.0
11                 1.500000                    47.368421            17.5
12                 1.166667                    87.500000            16.0
13                 0.333333                    15.384615            56.0
14                -0.833333                   -38.461538            32.0

    Valeur p
0   0.215732
1   0.008605
2   0.020977
3   0.005031
4   0.010151
5   0.475027
6   0.475027
7   0.126838
8   0.004655
9   0.004655
10  0.215732
11  0.003904
12  0.003904
13  0.215732
14  0.013011
```

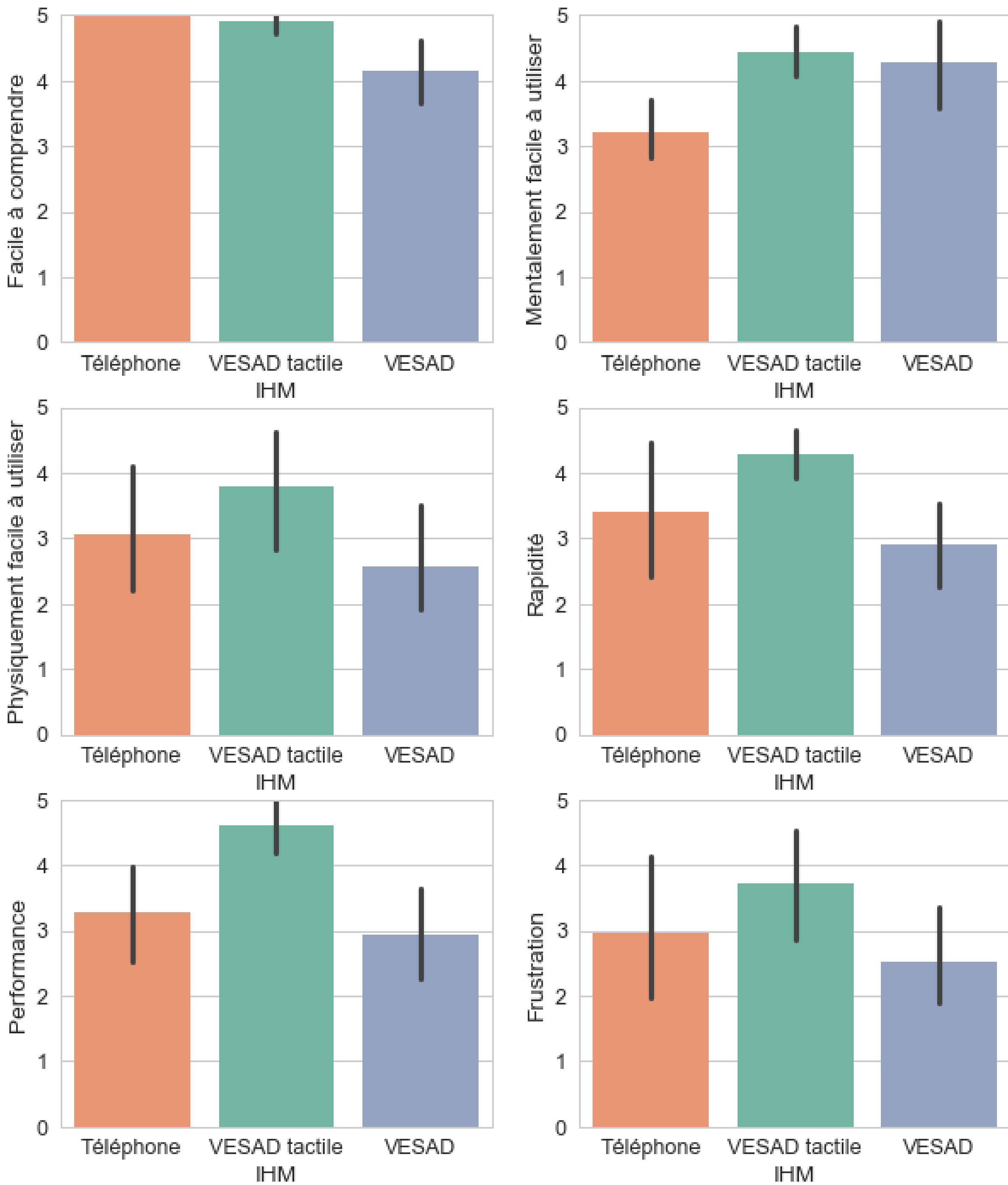
We display the geometric mean and 95% CI of each question:

```
In [35]: ranks.groupby([ivs['technique']['label']]).aggregate(print_geo_mean_ci)\
```

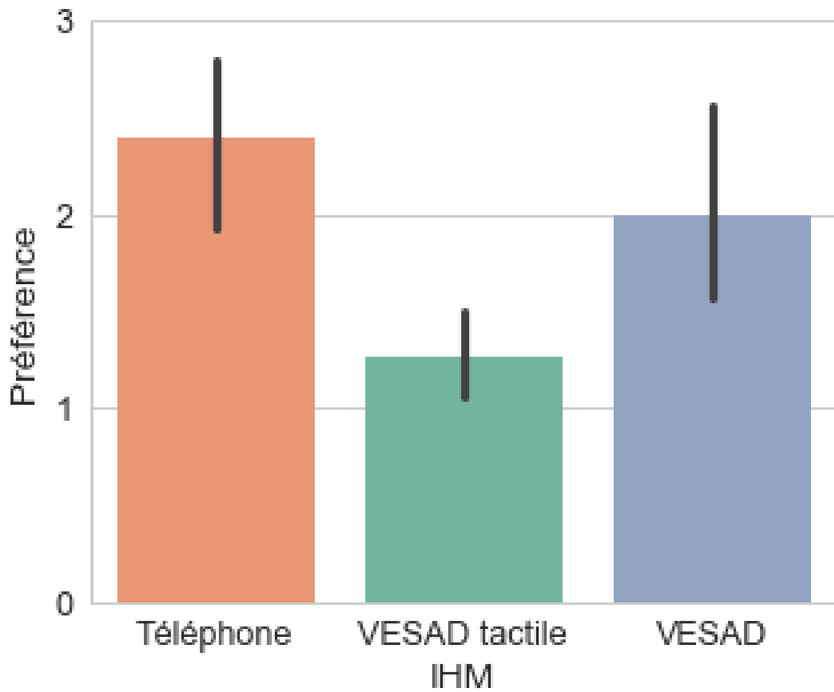
```
        .loc[:, ranks_dvs.loc['label', :]].transpose()
```

```
Out[35]: IHM                                    Téléphone      VESAD tactile  \
         Facile à comprendre               5.00 [5.00, 5.00]  4.91 [4.73, 5.00]
         Mentalement facile à utiliser     3.22 [2.77, 3.68]  4.45 [4.03, 4.82]
         Physiquement facile à utiliser    3.06 [2.16, 4.16]  3.80 [2.75, 4.62]
         Rapidité                          3.41 [2.42, 4.55]  4.29 [3.94, 4.64]
         Performance                       3.27 [2.51, 3.98]  4.62 [4.16, 5.00]
         Frustration                       2.96 [2.02, 4.14]  3.73 [2.71, 4.56]
         Préférence                        2.39 [1.95, 2.80]  1.26 [1.06, 1.50]

         IHM                                        VESAD
         Facile à comprendre               4.16 [3.73, 4.70]
         Mentalement facile à utiliser     4.28 [3.60, 4.91]
         Physiquement facile à utiliser    2.58 [1.88, 3.42]
         Rapidité                          2.90 [2.29, 3.51]
         Performance                       2.94 [2.27, 3.61]
         Frustration                       2.53 [1.88, 3.31]
         Préférence                        1.99 [1.55, 2.47]
```

```
In [36]: (fig, axs) = plot_ranks('technique', ranks_dvs.columns[0:-1],
                                 estimator=geometric_mean)
         fig.savefig('ranks.png')
```




```
In [37]: (fig, axs) = plot_ranks('technique', ['preference'],
                                 estimator=geometric_mean)
         fig.savefig('preferences.png')
```



Overall significant results are:

- **Easy to Understand**: *PhoneOnly* is significantly better than *MidAirInArOut* ($p = 0.009$), and seems a little better than *PhoneInArOut*.
- **Physically Easy to Use**: There is no significant differences due to TECHNIQUE; they seem scored similar.
- **Mentally Easy to Use**: *PhoneOnly* is statistically and practically worst than *PhoneInArOut* ($p = 0.005$) and *MidAirInArOut* ($p = 0.01$).
- **Subjective Speed**: *PhoneInArOut* is significantly better than *MidAirInArOut* ($p = 0.05$).
- **Subjective Performance**: *PhoneInArOut* is statistically better than *PhoneOnly* ($p = 0.005$) and *MidAirInArOut* ($p = 0.004$).
- **Frustration**: There is no significant differences due to TECHNIQUE; they seem scored similar.
- **Preference**: *PhoneInArOut* is significantly prefered to *PhoneOnly* ($p = 0.004$) and *MidAirInArOut* ($p = 0.01$).

0.3 3. Participant trials

Some functions for the analysis:

```
In [38]: def trial_samples(iv_id, dv_id, data=trials):
             iv, dv = ivs[iv_id], trials_dvs[dv_id]
             return [data[data[iv['label']] == iv_value][dv]\
                     for iv_value in iv['categorical']]
```

```
In [39]: def trial_means(iv_ids, dv_ids, data=trials, aggregate=print_mean_ci):
             iv_labels = [ivs.at['label', iv_id] for iv_id in iv_ids]
             dv_labels = [trials_dvs[dv_id] for dv_id in dv_ids]
             return data[iv_labels + dv_labels].groupby(iv_labels)\
                                               .aggregate(aggregate)
```

In [40]:
```python
def melt_trials(value_vars, var_name, value_name, data=trials):
    return pd.melt(data, id_vars=ivs.loc['label', :],
                   value_vars=value_vars, var_name=var_name,
                   value_name=value_name)
```

In [41]:
```python
def test_normality(iv_ids, dv_ids, data=trials, **args):
    results = []

    for dv_id in dv_ids:
        for iv_id in iv_ids:
            iv = ivs[iv_id]
            samples = trial_samples(iv_id, dv_id, data)
            for iv_value, sample in zip(iv['categorical'], samples):
                W, p = stats.shapiro(sample)
                results.append([iv['label'], iv_value, trials_dvs[dv_id], W, p])

    results = pd.DataFrame(results, columns=[labels['iv'], labels['iv_value'],
                                             labels['dv'], 'Shapiro W',
                                             labels['p_value']])
    p_values_correction(results, **args)
    return results
```

In [42]:
```python
def test_equal_variances(iv_ids, dv_ids, data=trials, levene_center='mean',
                         **args):
    results = []

    for dv_id in dv_ids:
        for iv_id in iv_ids:
            samples = trial_samples(iv_id, dv_id, data)
            W, p = stats.levene(*samples)
            results.append([ivs[iv_id]['label'], trials_dvs[dv_id], W, p])

    results = pd.DataFrame(results, columns=[labels['iv'], labels['dv'],
                                             'Levene W', labels['p_value']])
    p_values_correction(results, **args)
    return results
```

In [43]:
```python
def test_pairwise_trials(iv_id, dv_id, data=trials, log_data=False, **args):
    results = []
    iv, dv = ivs[iv_id], trials_dvs[dv_id]
    iv_category_ids = range(len(iv['categorical']))

    samples = trial_samples(iv_id, dv_id, data)
    sample_pairs = itertools.combinations(iv_category_ids, 2)
    for id1, id2 in sample_pairs:
        T, p = stats.ttest_ind(samples[id1], samples[id2])

        mean_diff, mean_diff_per = mean_difference(np.exp(samples[id1]),
                                                   np.exp(samples[id2]))\
            if log_data else mean_difference(samples[id1], samples[id2])
```

```
            results.append([iv['label'], iv['categorical'][id1],
                            iv['categorical'][id2], dv, mean_diff,
                            mean_diff_per, T, p])

    columns = [labels['iv'], labels['iv_value'] + ' 1',
               labels['iv_value'] + ' 2', labels['dv'],
               labels['mean_difference'], labels['mean_difference_percentage'],
               labels['t_statistic'], labels['p_value']]
    results = pd.DataFrame(results, columns=columns)

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

```
In [44]: def test_non_normal_trials(iv_ids, dv_ids, data=trials, **args):
             results = []

             for dv_id in dv_ids:
                 for iv_id in iv_ids:
                     samples = trial_samples(iv_id, dv_id, data)
                     H, p = stats.kruskal(*samples)
                     results.append([ivs[iv_id]['label'], trials_dvs[dv_id], H, p])

             results = pd.DataFrame(results, columns=[labels['iv'], labels['dv'],
                                                      'Kruskal-Wallis H',
                                                      labels['p_value']])
             p_values_correction(results, **args)
             return results
```

```
In [45]: def test_pairwise_non_normal_trials(iv_ids, dv_ids, data=trials, **args):
             results = []

             for dv_id in dv_ids:
                 for iv_id in iv_ids:
                     iv, dv = ivs[iv_id], trials_dvs[dv_id]
                     samples = trial_samples(iv_id, dv_id, data)

                     iv_category_ids = range(len(iv['categorical']))
                     sample_pairs = itertools.combinations(iv_category_ids, 2)
                     for id1, id2 in sample_pairs:
                         U, p = stats.mannwhitneyu(samples[id1], samples[id2])
                         mean_diff, mean_diff_per = mean_difference(samples[id1],
                                                                    samples[id2])
                         results.append([iv['label'], iv['categorical'][id1],
                                         iv['categorical'][id2], dv, mean_diff,
                                         mean_diff_per, U, p])

             columns = [labels['iv'], labels['iv_value'] + ' 1',
                        labels['iv_value'] + ' 2', labels['dv'],
                        labels['mean_difference'], labels['mean_difference_percentage'],
                        'Mann-Whitney U', labels['p_value']]
             results = pd.DataFrame(results, columns=columns)
```

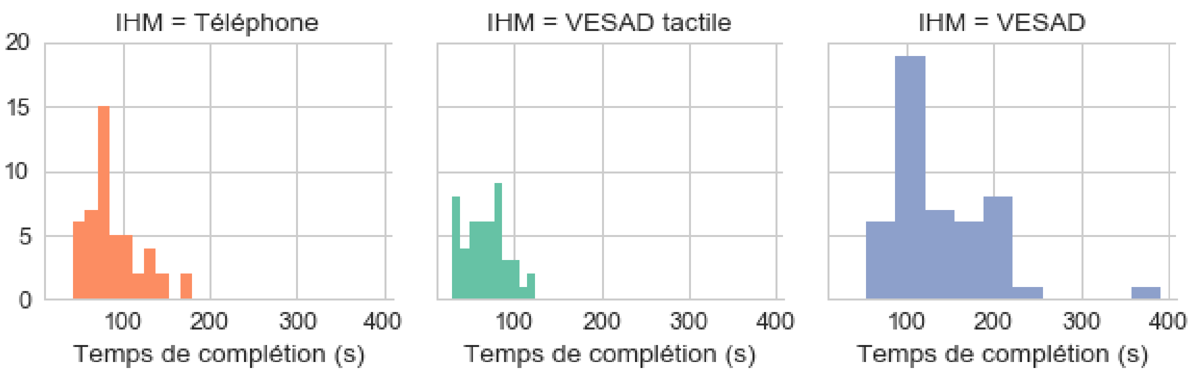
```
        p_values_correction(results, **args)
        return results
```

```
In [46]: def plot_trials(iv_ids_list, dv_id, data=trials, kind='bar', **args):
             dv = trials_dvs[dv_id]
             if (len(iv_ids_list) == 0):
                 iv_ids_list = [[iv_id] for id_id in ivs.columns]

             fig, axs = subplots(len(iv_ids_list))
             for id_ids, ax in zip(iv_ids_list, axs):

                 iv_ids = [ivs[id_id] for id_id in id_ids]
                 if (len(iv_ids) == 1):
                     if (kind == 'bar'):
                         sns.barplot(x=iv_ids[0]['label'], y=dv, data=data,
                                     palette=iv_ids[0]['palette'], ax=ax, **args)

                     elif (kind == 'box'):
                         sns.boxplot(x=iv_ids[0]['label'], y=dv, data=data,
                                     palette=iv_ids[0]['palette'], ax=ax, **args)

                     elif (kind == 'count'):
                         sns.countplot(hue=iv_ids[0]['label'], x=dv, data=data,
                                       palette=iv_ids[0]['palette'], ax=ax, **args)
                         ax.set(ylabel='Count')
                         ax.legend(loc='upper right', title=labels['count'],
                                   frameon=True)

                 elif (len(ivs) == 2):
                     if (kind == 'bar'):
                         sns.barplot(x=iv_ids[1]['label'], y=dv, hue=iv_ids[0]['label'],
                                     data=data, palette=iv_ids[0]['palette'], ax=ax,
                                     **args)
                     ax.legend(frameon=True, loc='upper left', bbox_to_anchor=(1, 1))

             return (fig, axs)
```

0.3.1 3.1. Task completion time

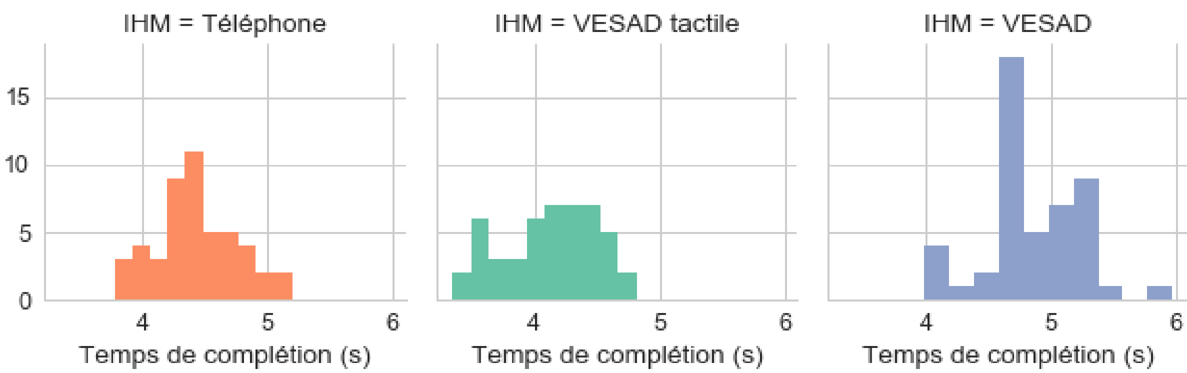
We first apply a log transform to TCT to approximate a normal distribution.

```
In [47]: g = sns.FacetGrid(trials, col=technique['label'], hue=technique['label'],
                           palette=technique['palette'])
         g = g.map(plt.hist, total_time)
         g.savefig('tct_distributions.png')
```



In [48]:
```
trials[total_time] = np.log(trials[total_time])
raw_trials['total_time'] = np.log(raw_trials['total_time'])
```

In [49]:
```
g = sns.FacetGrid(trials, col=technique['label'], hue=technique['label'],
                  palette=technique['palette'])
g = g.map(plt.hist, total_time)
g.savefig('tct_distributions_log.png')
```



We test the normality of TCT distributions for each TECHNIQUE and their equality of variances, since it's the main factor of interest.

In [50]:
```
test_normality(['technique'], ['total_time'])
```

Out[50]:

	VI	Valeur VI	VD	Shapiro W	Valeur p
0	IHM	Téléphone	Temps de complétion (s)	0.983679	0.735995
1	IHM	VESAD tactile	Temps de complétion (s)	0.971828	0.446912
2	IHM	VESAD	Temps de complétion (s)	0.966510	0.446912

In [51]:
```
test_equal_variances(['technique'], ['total_time'])
```

Out[51]:

	VI	VD	Levene W	Valeur p
0	IHM	Temps de complétion (s)	0.756269	0.471309

We meet all the assumptions of an ANOVA. Trials were done independently, TCT distributions are normal and their variances are equal.

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

```
In [52]: tct_model = ols('total_time ~ technique * text_size * distance'
                         + '+ technique * ordering', data=raw_trials).fit()
         sm.stats.anova_lm(tct_model, typ=2)
```

Out[52]:

	sum_sq	df	F	PR(>F)
technique	12.328279	2.0	62.210952	1.610604e-19
text_size	0.145175	1.0	1.465169	2.283752e-01
distance	0.029459	1.0	0.297314	5.865353e-01
ordering	2.133256	2.0	10.764832	4.829976e-05
technique:text_size	0.398420	2.0	2.010508	1.381943e-01
technique:distance	0.074854	2.0	0.377730	6.861892e-01
text_size:distance	0.291976	1.0	2.946742	8.850822e-02
technique:ordering	3.086868	4.0	7.788476	1.215733e-05
technique:text_size:distance	0.499563	2.0	2.520896	8.443561e-02
Residual	12.484644	126.0	NaN	NaN

The main significant effect on TCT is TECHNIQUE ($F = 62.2$, $p < 0.0001$), then ORDERING ($F = 10.8$, $p < 0.0001$). There is also an significant interaction effect: TECHNIQUE x ORDERING ($F = 2.5$, $p < 0.0001$).

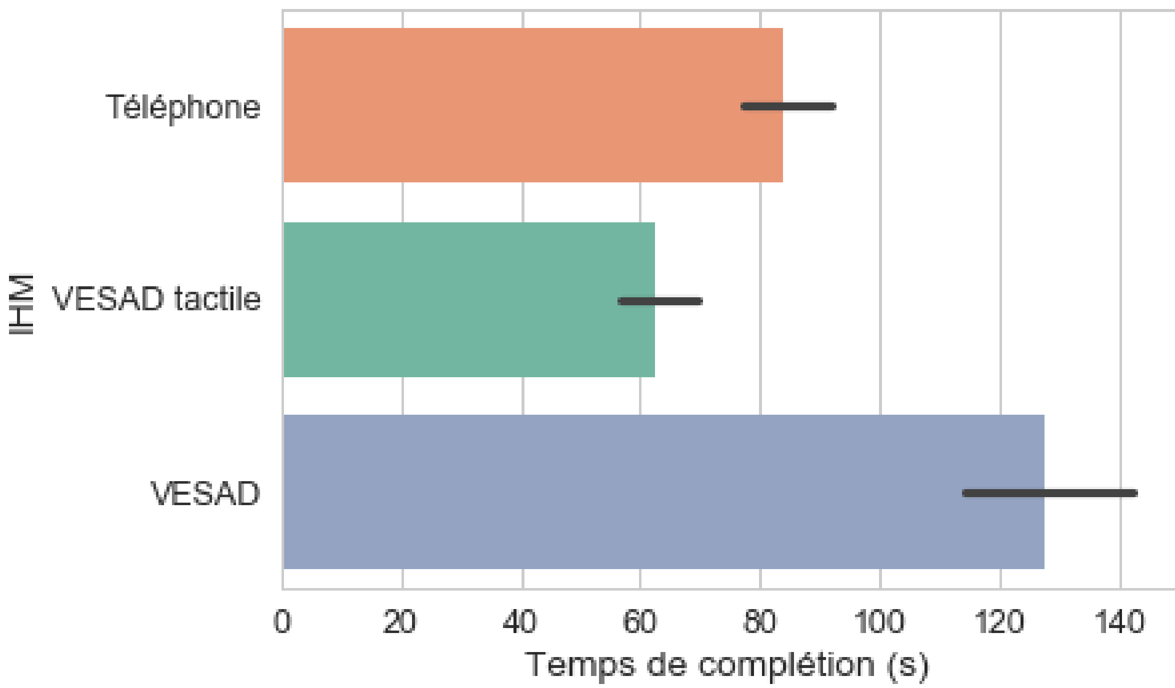
We display mean TCT values with 95% CI for these conditions:

```
In [53]: trial_means(['technique'], ['total_time'], aggregate=print_exp_mean_ci)
```

Out[53]:

IHM	Temps de complétion (s)
Téléphone	84.02 [76.81, 92.76]
VESAD tactile	62.59 [56.06, 69.33]
VESAD	127.71 [114.14, 141.96]

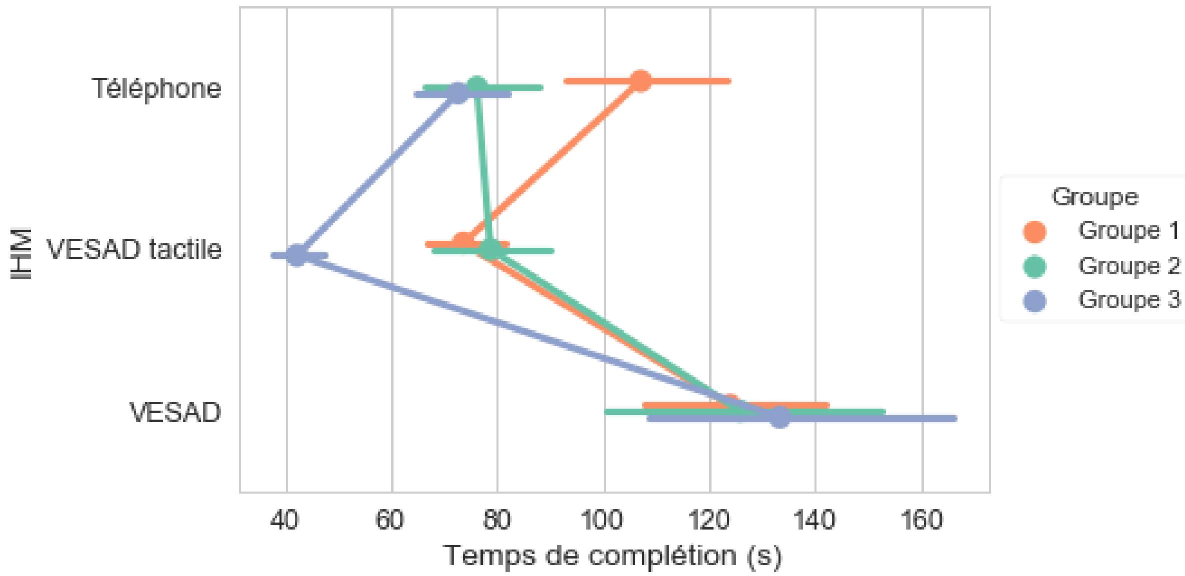
```
In [54]: ax = sns.barplot(x=trials_dvs['total_time'], y=technique['label'],
                          palette=technique['palette'], data=trials, estimator=exp_mean)
         ax.get_figure().savefig('tct.png')
```




```
In [55]: trial_means(['ordering', 'technique'], ['total_time'],
                     aggregate=print_exp_mean_ci).unstack()
```

```
Out[55]:          Temps de complétion (s)                           \
         IHM                    Téléphone         VESAD tactile
         Groupe
         Groupe 1  107.14 [92.23, 123.88]  73.63 [67.26, 81.12]
         Groupe 2    76.22 [66.31, 87.24]  78.79 [68.15, 89.81]
         Groupe 3    72.64 [64.96, 82.05]  42.27 [37.99, 47.55]


         IHM                          VESAD
         Groupe
         Groupe 1  124.00 [108.09, 141.20]
         Groupe 2  125.91 [101.42, 152.20]
         Groupe 3  133.41 [108.95, 169.41]
```

```
In [56]: ax = sns.pointplot(x=total_time, y=technique['label'], hue=ordering['label'],
                            palette=ordering['palette'], data=trials, dodge=True,
                            estimator=exp_mean)
         config_legend(ax, 'ordering')
         ax.get_figure().savefig('tct_ordering.png')
```



It seems that participants who started with *PhoneOnly* were slower with this technique than the the other groups. Similarly, participants who finished with *PhoneInArOut* were faster with this technique. It indicates there is a learning curve on the task, but interestingly participants from all groups performed equally with *MidAirInArOut* technique.

We compare the TCT for the three techniques only with pairwise t-tests (Benjamini–Hochberg correction) .

```
In [57]: test_pairwise_trials('technique', 'total_time', correction_method='fdr_bh',
                              log_data=True)
```

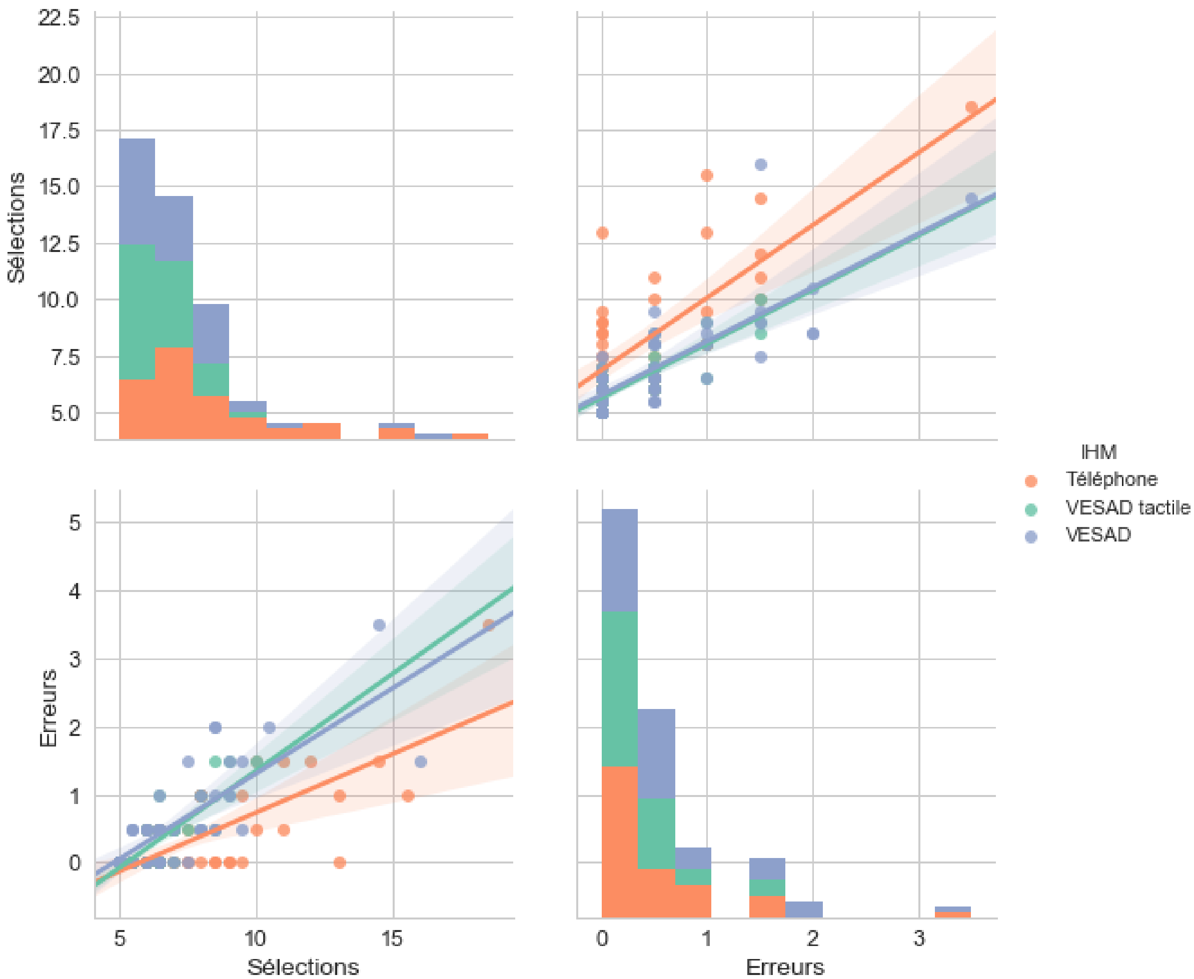
```
Out[57]:    VI    Valeur VI 1    Valeur VI 2                       VD  \
         0  IHM      Téléphone  VESAD tactile  Temps de complétion (s)
         1  IHM      Téléphone          VESAD  Temps de complétion (s)
         2  IHM  VESAD tactile          VESAD  Temps de complétion (s)

            Différence des moyennes  Différence des moyennes (%)  Statistique T  \
         0                21.840362                    32.666819       4.090635
         1               -49.467731                   -35.803106      -5.615875
         2               -71.308093                   -51.610437      -9.032751

                Valeur p
         0  9.077859e-05
         1  2.966612e-07
         2  6.282833e-14
```

- *PhoneInArOut* is 71s (+52%) faster than *MidAirInArOut* ($p < 0.0001$).
- *PhoneOnly* is 49s (+36%) faster than *MidAirInArOut* ($p < 0.0001$).
- *PhoneInArOut* is 22s (+33%) faster than *PhoneOnly* ($p < 0.0001$).

0.3.2 3.2. Errors

We visualize first the SELECTIONS and ERRORS distributions:

```
In [58]: g = sns.pairplot(trials, hue=technique['label'], kind='reg',
                          vars=[trials_dvs['selections_count'], trials_dvs['errors']],
                          palette=technique['palette'], size=4);
         g.savefig('selections_errors_distributions.png')
```



It seems that a user makes as much errors as she makes selections. The relation is almost the same for each technique, even if users seems to make more selections for the same number of errors with PhoneOnly.

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

```
In [59]: test_non_normal_trials(['technique', 'text_size', 'distance', 'ordering'],
                                ['selections_count', 'errors'])
```

Out[59]:

	VI	VD	Kruskal-Wallis H	Valeur p
0	IHM	Sélections	15.125918	0.004155
1	Taille du texte	Sélections	0.003897	0.950224
2	Distance	Sélections	0.052701	0.935347
3	Groupe	Sélections	11.510535	0.010095

```
4              IHM     Erreurs          5.201541  0.148433
5  Taille du texte     Erreurs          0.103661  0.935347
6         Distance     Erreurs          0.201599  0.935347
7           Groupe     Erreurs         11.153096  0.010095
```

Only TECHNIQUE ($p = 0.004$) and ORDERING ($p = 0.01$) have a significant effect on SELECTIONS. But, only ORDERING ($p = 0.01$) have a significant effect on ERRORS.

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

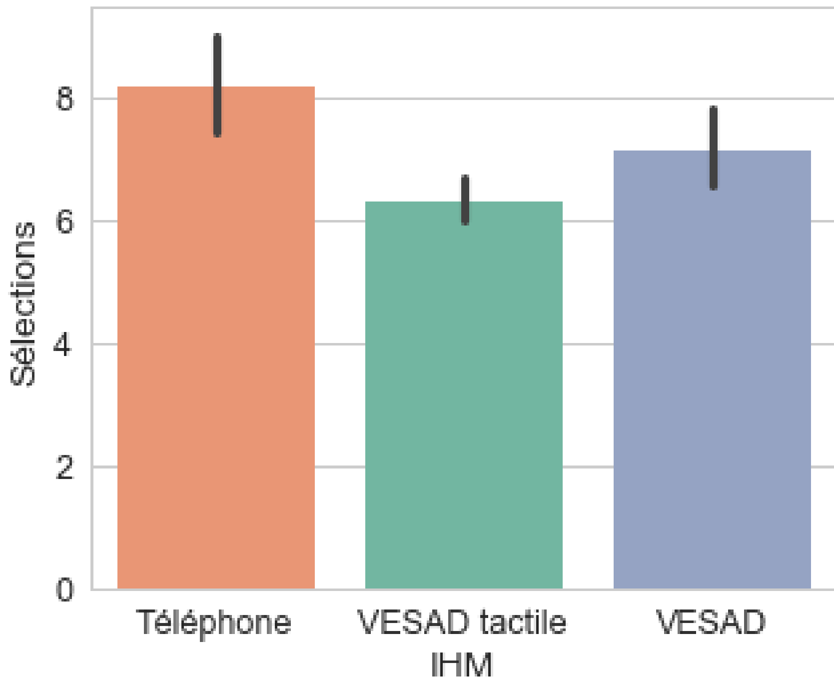
```
In [60]: test_pairwise_non_normal_trials(['technique', 'ordering'],
                                          ['selections_count', 'errors'])
```

```
Out[60]:         VI     Valeur VI 1    Valeur VI 2          VD  Différence des moyennes  \
         0      IHM       Téléphone  VESAD tactile  Sélections                 1.864583
         1      IHM       Téléphone          VESAD  Sélections                 1.041667
         2      IHM   VESAD tactile          VESAD  Sélections                -0.822917
         3   Groupe         Groupe 1       Groupe 2  Sélections                 1.729167
         4   Groupe         Groupe 1       Groupe 3  Sélections                 1.583333
         5   Groupe         Groupe 2       Groupe 3  Sélections                -0.145833
         6      IHM       Téléphone  VESAD tactile     Erreurs                 0.125000
         7      IHM       Téléphone          VESAD     Erreurs                -0.166667
         8      IHM   VESAD tactile          VESAD     Erreurs                -0.291667
         9   Groupe         Groupe 1       Groupe 2     Erreurs                 0.333333
         10  Groupe         Groupe 1       Groupe 3     Erreurs                 0.343750
         11  Groupe         Groupe 2       Groupe 3     Erreurs                 0.010417

             Différence des moyennes (%)  Mann-Whitney U  Valeur p
         0                     29.537954           627.0  0.000635
         1                     14.598540           864.5  0.029356
         2                    -11.532847           915.0  0.059881
         3                     26.265823           755.0  0.005196
         4                     23.529412           755.0  0.005196
         5                     -2.167183          1136.0  0.454285
         6                     42.857143          1083.5  0.312007
         7                    -28.571429           957.0  0.083050
         8                    -50.000000           873.0  0.026306
         9                    103.225806           846.5  0.020183
         10                   110.000000           761.5  0.005196
         11                     3.333333          1039.0  0.205336
```

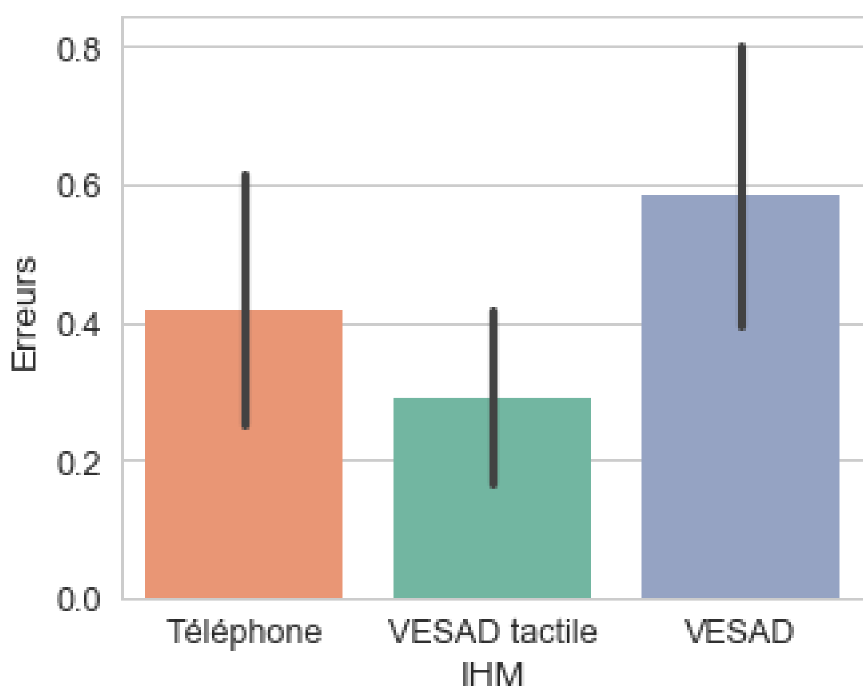
```
In [61]: trial_means(['technique'], ['selections_count', 'errors'])
```

```
Out[61]:                       Sélections              Erreurs
         IHM
         Téléphone      8.18 [7.42, 8.99]  0.42 [0.24, 0.59]
         VESAD tactile  6.31 [5.97, 6.70]  0.29 [0.18, 0.42]
         VESAD          7.14 [6.54, 7.86]  0.58 [0.39, 0.80]
```

```
In [62]: (fig, axs) = plot_trials([['technique']], 'selections_count')
         fig.savefig('selections.png')
```



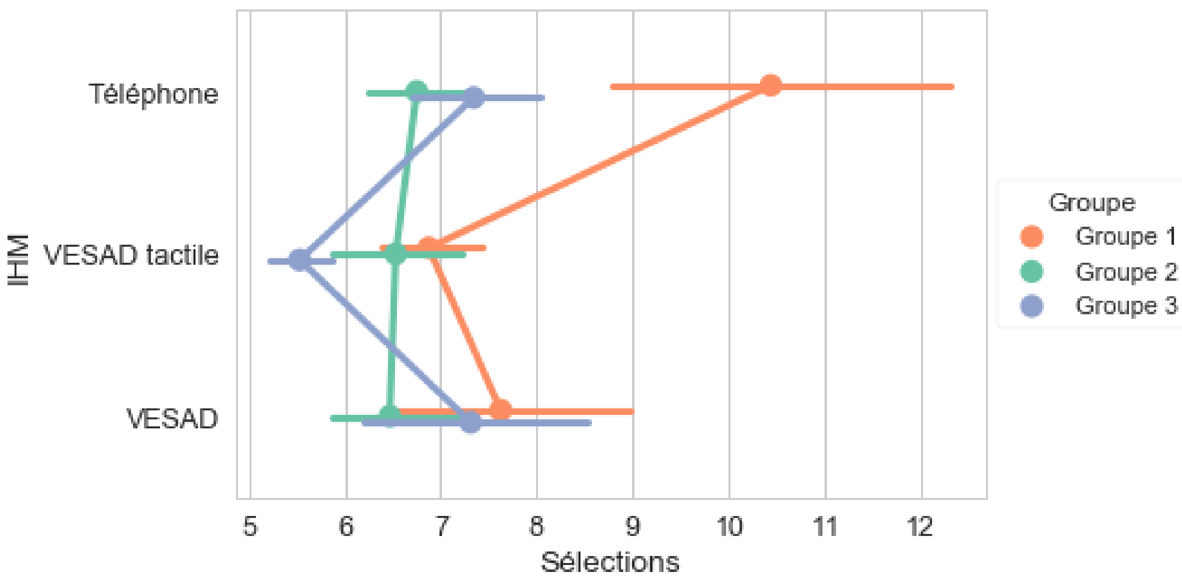

```
In [63]: (fig, axs) = plot_trials([['technique']], 'errors')
         fig.savefig('errors.png')
```



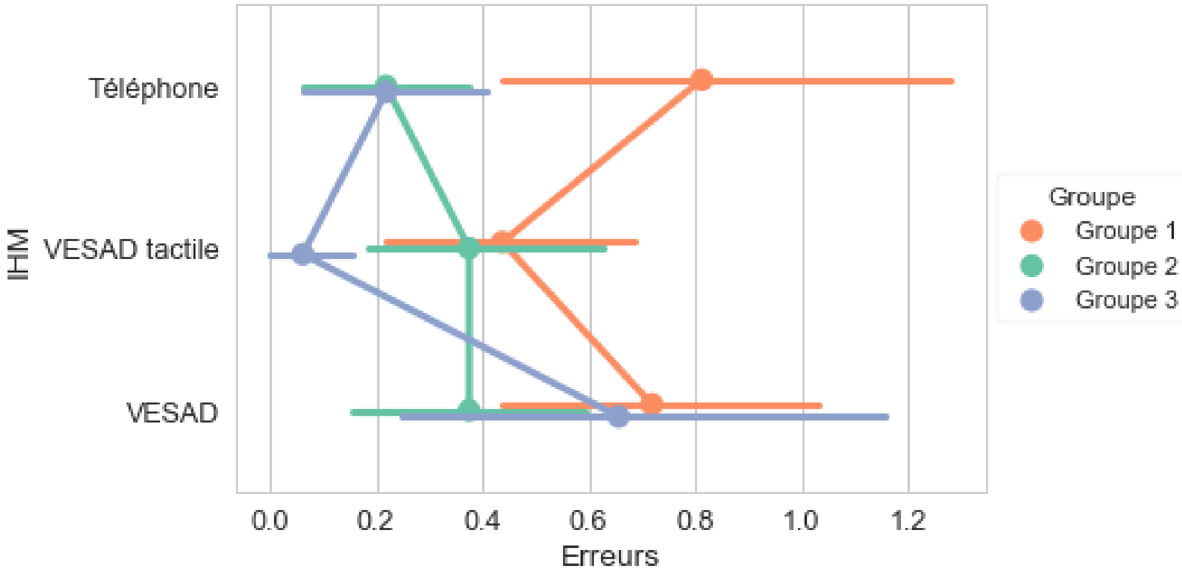

```
In [64]: trial_means(['ordering', 'technique'], ['selections_count', 'errors'])
```

Out[64]:
```
                                Sélections           Erreurs
Groupe   IHM
Groupe 1 Téléphone      10.44 [8.72, 12.31]  0.81 [0.38, 1.28]
         VESAD tactile   6.88 [6.38, 7.50]   0.44 [0.22, 0.69]
         VESAD           7.62 [6.62, 9.12]   0.72 [0.41, 1.06]
Groupe 2 Téléphone       6.75 [6.25, 7.31]   0.22 [0.06, 0.41]
         VESAD tactile   6.53 [5.87, 7.22]   0.38 [0.16, 0.59]
         VESAD           6.47 [5.91, 7.13]   0.38 [0.19, 0.62]
Groupe 3 Téléphone       7.34 [6.72, 8.06]   0.22 [0.06, 0.38]
         VESAD tactile   5.53 [5.25, 5.84]   0.06 [0.00, 0.16]
         VESAD           7.31 [6.28, 8.69]   0.66 [0.25, 1.13]
```

In [65]:
```python
ax = sns.pointplot(x=trials_dvs['selections_count'], y=technique['label'],
                   hue=ordering['label'], palette=ordering['palette'],
                   data=trials, dodge=True)
config_legend(ax, 'ordering')
ax.get_figure().savefig('selections_ordering.png')
```



In [66]:
```python
ax = sns.pointplot(x=trials_dvs['errors'], y=technique['label'],
                   hue=ordering['label'], palette=ordering['palette'],
                   data=trials, dodge=True)
config_legend(ax, 'ordering')
ax.get_figure().savefig('errors_ordering.png')
```



0.3.3 3.3. Navigation

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

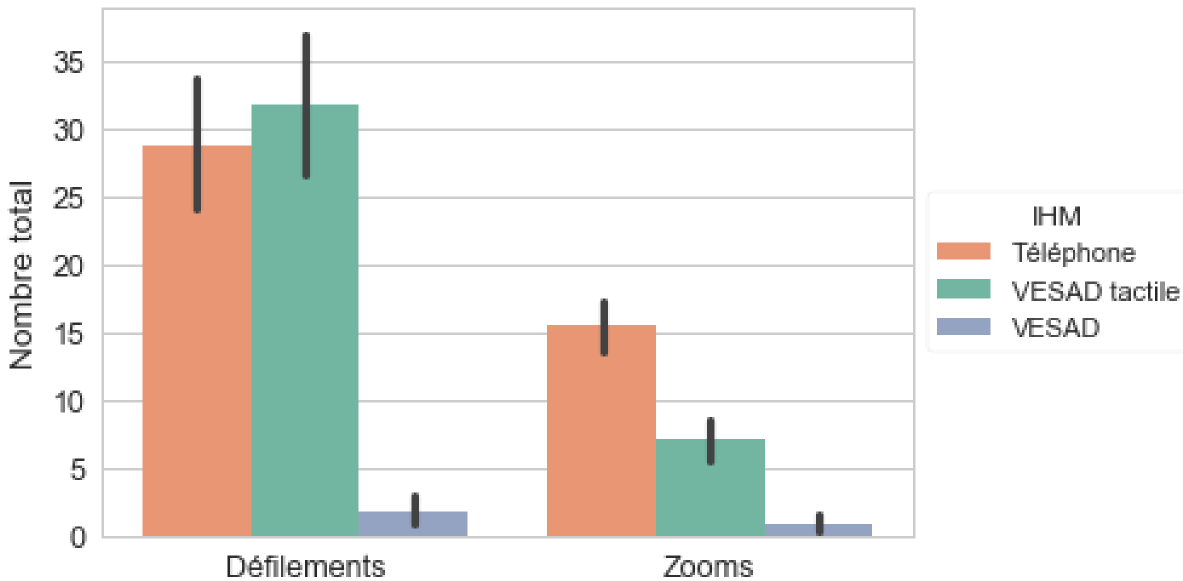
In [67]:
```
# Data preparation
trial_counts = melt_trials(var_name=labels['category'],
                           value_name=labels['count'],
                           value_vars=[trials_dvs['pan_count'],
                                       trials_dvs['zoom_count']])

trial_times = melt_trials(var_name=labels['category'],
                          value_name=labels['time'],
                          value_vars=[trials_dvs['selections_time'],
                                      trials_dvs['pan_time'],
                                      trials_dvs['zoom_time']])

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

In [68]:
```
trial_means(['technique'], ['pan_count', 'zoom_count'])
```

Out[68]:
```
                        Défilements                   Zooms
IHM
```

```
Téléphone      28.70 [24.34, 33.28]  15.53 [13.62, 17.53]
VESAD tactile  31.81 [26.62, 37.60]     7.14 [5.72, 8.81]
VESAD             1.83 [0.81, 3.08]     0.83 [0.27, 1.53]
```

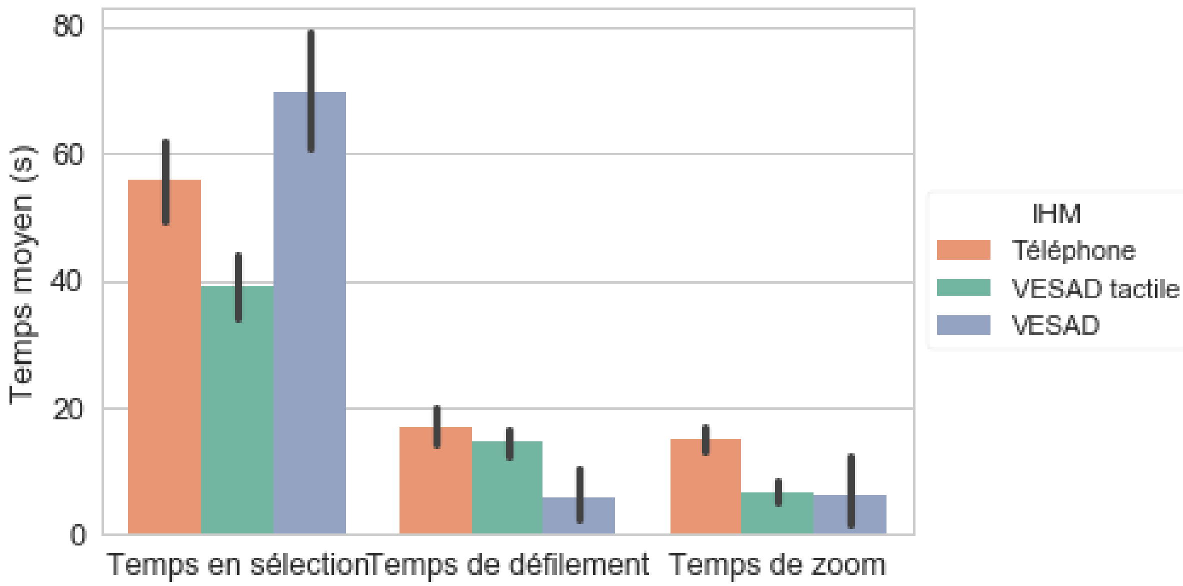
In [69]:
```python
ax = sns.barplot(x=labels['category'], y=labels['count'],
                 hue=technique['label'], palette=technique['palette'],
                 data=trial_counts)
config_legend(ax, 'technique')
ax.set(xlabel='')
ax.get_figure().savefig('navigation_count.png')
```



In [70]:
```python
trial_means(['technique'], ['selections_time', 'pan_time', 'zoom_time'])
```

Out[70]:
```
                   Temps en sélection   Temps de défilement  \
IHM
Téléphone      55.74 [50.20, 61.79]  17.12 [14.48, 19.97]
VESAD tactile  38.98 [34.32, 43.80]  14.53 [12.51, 16.50]
VESAD          69.68 [60.35, 79.78]    5.97 [2.15, 10.07]

                     Temps de zoom
IHM
Téléphone      15.18 [13.31, 17.18]
VESAD tactile     6.70 [5.11, 8.62]
VESAD            6.24 [1.67, 11.87]
```

In [71]:
```python
ax = sns.barplot(x=labels['category'], y=labels['time'], data=trial_times,
                 hue=technique['label'], palette=technique['palette'])
config_legend(ax, 'technique')
ax.set(xlabel='')
ax.get_figure().savefig('navigation_time.png')
```




```
In [72]: trial_means(['technique'], ['selections_projected_distance',
                                     'pan_projected_distance',
                                     'zoom_projected_distance',
                                     'absolute_head_phone_distance'])

Out[72]:                Distance en sélection Distance de défilement   Distance de zoom  \
         IHM
         Téléphone        4.87 [4.09, 5.68]      1.50 [1.20, 1.82]  2.11 [1.75, 2.55]
         VESAD tactile    2.83 [2.40, 3.28]      1.09 [0.91, 1.29]  0.69 [0.51, 0.89]
         VESAD           8.72 [7.19, 10.53]      0.47 [0.18, 0.86]  0.62 [0.15, 1.19]

                        Mouvements tête-téléphone
         IHM
         Téléphone              3.18 [2.34, 4.12]
         VESAD tactile          1.57 [1.27, 1.91]
         VESAD                  6.12 [4.93, 7.43]

In [73]: g = sns.factorplot(x=technique['label'], y=labels['distance'],
                            col=labels['category'], data=trial_distances,
                            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=technique['label'])
fix_legend_fontsize(legend)

g.savefig('navigation_distance.png')
```