RETTL Project Summer 2022 Sprint 1 Intern Report: Multimodal Classroom Data Mining and Triangulation

Tianze (Steven) Shou

7/5/2022

Abstract

Between the three days of May 23 and May 25, 2022, data from three modalities (teacher position, observation log, and student tutor log) have been recorded for 5 individual class periods. During this sprint, the focus is on understanding, decomposing and distilling information from position data and observation log data and assessing their level of inter-modal alignment, or so called multimodal triangulation. The outcome of data mining from these two modalities will follow the event-actor-subject format, where for each data point, an event name (e.g., talking to student), an actor (e.g., the teacher), and an optional subject (e.g., student #8) will be specified. In the triangulation process, the observation log data is used as a benchmark to tune he parameter involved in data mining the position data. A reward-penalty system is devised as a indicator is inter-modal alignment together with five percentages accessing the overlapping between events generated from position data and the observation data.

Introduction

Context of Data Collection

The multimodal data is collected from an eastern Pennsylvania middle school during consecutive 3 days amongst 5 class periods with different groups of students (15 class session in total). All classroom objects like seat arrangements and teacher desk are located in the same position for the entire duration of the 3-day data collection period. Student seatings are designed to be the same for each individual period, but some students do change seats before the period begins due to technical or disciplinary causes. These changes are well-documented as they closely relate to the later spatial pedagogy analysis. Few seat change takes place during class period, but is documented as well. Figure 1 is a overview of classroom layout. The more specific coordinates of classroom objects are in appendix I.

Data Collection Methods

The teacher's position data is collected from the Pozyx position sensing system, a commercial integration that is commonly used in the field of spatial pedagogy. The classroom is equipped with six `anchors` installed to the walls, close to the ceiling. The teacher wears two position sensing tags, one hanging in front of the neck, the other on the back. The anchors collect position information from the two tag every second and output a new row of log to the raw data log file with unix time stamp, X and Y coordinates, and confidence score in the coordinates ranging from 0 to 100.

The observation log data is manually coded by Shamya Karumbaiah according to the coding scheme specified in Figure 2. Unfortunately, only one coder is involved in this process, which is also why cross-validation between position data and observation data is necessary. Most of the events observed will be coded in `action` + `where` (optional) + `student ID` (optional) + `keyword` (optional) format. Actor-subject format data will also be distilled from this coding later.

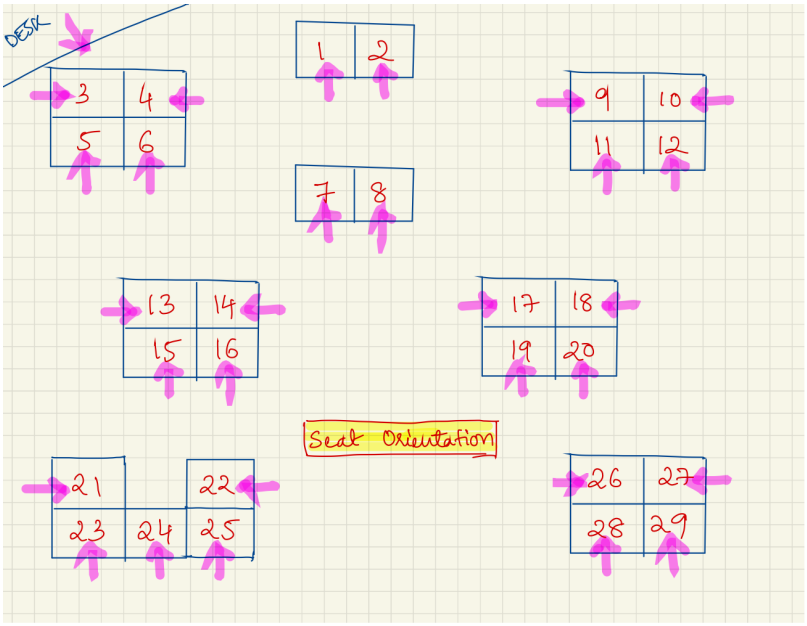

Figure 1: Classroom Layout

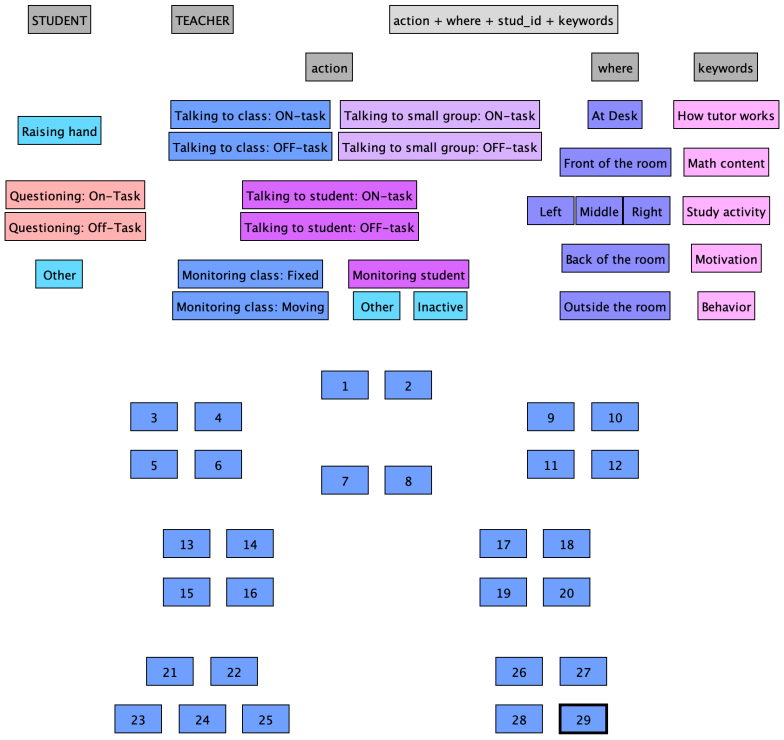

Figure 2: Coding Scheme

Tutor log data is collected from the MathTutor platform that is built upon the Lynett system. The tutor data have already been collected as a form of students' input, but the decomposion and analysis of this modality is out-of-scope for this sprint, and will be discussed in the next sprint.

Position Data Mining

Since position data is collected from two tags on the same individual, there exists some dispute between the two position data sources. This is resolved by introducing the confidence score from Pozyx log. The chosen X and Y coordinates are essentially the weighted average between the two coordinates returned by the two tags. How it is done mathematically is specified in the formula:

$$[X, Y] = \frac{c_1[X_1, Y_1] + c_2[X_2, Y_2]}{c_1 + c_2}$$

where $[X_1, Y_1]$ and $[X_2, Y_2]$ are the coordinates reported by the two tags and c_1 and c_2 refers to their respective confidence score reported by the Pozyx system, and $[X, Y]$ is the final chosen coordinates.

Below is a tiny snippet of the distilled position data file:

```
##   time_stamp tag19_X tag19_Y tag20_X tag20_Y tag19_score tag20_score  chosen_X
## 1 1653308809   -1020    4514     -24    3608    89.43647    90.60095 -518.7789
## 2 1653308810      66    5563     244    4317    81.46247    89.49101  159.1797
## 3 1653308811    -537    5488     718    4541    86.76052    93.13711  112.7421
## 4 1653308812      47    5911    1545    5042    91.73187    90.81516  792.2387
## 5 1653308813     -29    6266    1933    4930    94.49976    85.03171  900.2645
## 6 1653308814     -29    6266    1933    4930    89.96525    84.98330  924.0644
##   chosen_Y periodID dayID
## 1 4058.070        1     1
## 2 4910.742        1     1
## 3 4997.716        1     1
## 4 5478.682        1     1
## 5 5633.229        1     1
## 6 5617.022        1     1
```

Stop Detection

Martínez-Maldonado et al. proposed the concept of *teacher's stops* based on teacher's position data [1]. The definition of `stop` involves two parameters: `duration` and `radius`. A `stop` can be established if the teacher's detected position remains within the distance, specified by `radius`, from the centroid point for no shorter than a continuous amount of time specified by `duration`. The centroid point is defined by the mean of all of the coordinate vectors. Figure 3 gives two situations where a `stop` is or is not established. All of the black crosses represent teacher position in consecutive ten seconds, and the blue dots are the centroids of these ten second's coordinates. In the example to the left, a `stop` is constituted since all of the black crosses (teacher's actual locations in the ten seconds) are with in the `radius distance`, whereas it is not the case in the example to the right.

Observation Data Mining

Continuing on the discussion on the coding process of the observation log, it involves the coder clicking on the coding dashboard shown in Figure 2, and each click corresponds to an entry/row in the observation log. The automated process in data mining the observation log encapsulates the following heuristics:

- Catching on an entry with event name and create a new row in the output data file with this event
- Take the time stamp of this entry as the time stamp of this event

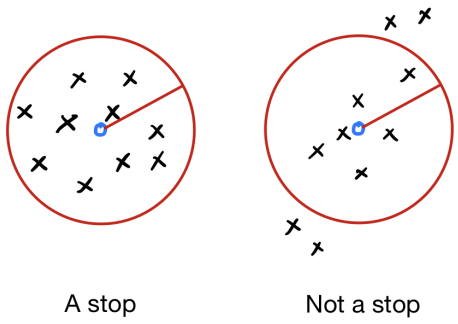

Figure 3: Stopping detection examples

- Determining whether it is teacher-conducted or student-conducted event (these concepts' explanation will follow)
- Picking up `where`, `keyword`, and `subject` in the following entries in raw log file, but these values may be NaN's
- Keep going through the entries in the log file, and repeat the aforementioned steps

The observation events can be either teacher-conducted or student-conducted:

- Teacher-conducted events are {*talking to student: on-task*, *talking to student: off-task*, *talking to class: on-task*, *talking to class: off-task*, *talking to small group: on-task*, *talking to small group: off-task*, *monitoring class: moving*, *monitoring class: fixed*, *monitoring student*, *inactive*}. These events have the teacher as the `actor`. Their `subject` can be a specific student, the entire class, a group of students, or NaN.
- Student-conducted events are {*raising hand*, *questioning: on-task*, *questioning: off-task*}. They have one of the students as `actor` and teacher as `subject`.

Here is a snippet of the distilled observation data:

```
## # A tibble: 6 x 10
##   periodID dayID   timestamp time   event      actor subject note  where keyword
##      <dbl> <dbl>       <dbl> <time> <chr>      <chr> <chr>   <chr> <chr> <chr>
## 1        1     1 1653308821  00:00  Period be~ <NA>  <NA>    <NA>  <NA>  <NA>
## 2        1     1 1653308839. 00:17  Talking t~ teac~ class   Watc~ <NA>  <NA>
## 3        1     1 1653308874. 00:52  Monitorin~ teac~ class   <NA>  <NA>  <NA>
## 4        1     1 1653308892. 01:11  Monitorin~ teac~ class   <NA>  <NA>  <NA>
## 5        1     1 1653308926. 01:45  Talking t~ teac~ 10      <NA>  <NA>  <NA>
## 6        1     1 1653308935. 01:53  Talking t~ teac~ 16      Just~ <NA>  <NA>
```

Multimodal Triangulation

As mentioned in the previous sections, the two parameters `duration` and `radius` are used to data mine `stops` from the position data generated by Pozyx system. Since inferring the subject of the teacher's visit during a particular `stop` is also of interests of this study, another parameter `range` is introduced.

`Range` is often confused with `radius`. The latter pertains to the establishment of `stops` while the earlier arbitrates the inference of classroom object that the teacher is next to during the `stop`. When a `stop` is established, the algorithm looks for students within the distance of `range` from the `stop` centroid and tag

this `stop` with this set of students as `potential subjects`. This set of students may have variable size, where it may be empty or may contain several students depending on the `range` parameter.

To see which combinations of the three parameters (`duration`, `radius`, and `range`) can help us best data mine the position data, a parameter sweep is carried out. With each combination, the `stops` and the `potential subjcets` will be compared with the observation data to see how much does the position data agree with the observation data. Two sets of rules explained in the proceeding sub-sections serve as the "loss function" as a metric of the level of agreement.

Scoring System

Since the observation data is generated by human coder, the inevitable lag in coding process must be addressed. This is why a `timeframe` is introduced. The scoring system starts with scanning through the observation data. If a teacher-stopping is seen (i.e., *talking to student: off-task*, *talking to student: on-task*, *talking to small group: off-task*, *talking to small group: on-task*), a timeframe, whose center is the time stamp of the observation event, will be created in the position data to conduct scoring. For instance, if there is a *talking to student: on-task* event at $t = 100$ and `timeframe` is set to be 20 seconds. We will only take the position data between $t = 100 - 20/2 = 90$ and $t = 100 + 20/2 = 110$ to scoring. Since the timeframe is 20 seconds, the position data usually yield 20 data points to scoring system. Specific scoring rules are the following:

- Each occurrence of row where teacher is not detected to be stopping will be penalized
- Each occurrence of row where teacher stops, but `potential subjects` is empty will be penalized
- Each occurrence of row where teacher stops, but `potential subjects` contains a false subject will be penalized for each false subject
- Each occurrence of row where teacher stops, and `potential subjects` contains the correct subject will be rewarded
- Any row indicating teacher's stops that is not in the timeframe of any observation events will be penalized

For now `timeframe` is set to be 20 seconds, and `penalty` is 1 point, `reward` 2 points. The following are the parameter combinations with the highest scores:

```
## # A tibble: 6 x 10
##   duration radius range  score perc_of_stops_in_both right_match_percentage
##      <dbl>  <dbl> <dbl>  <dbl>                 <dbl>                  <dbl>
## 1       27    200   700  -9863                 0.147                  0.394
## 2       27    200   900  -9989                 0.147                  0.535
## 3       19    200   700 -10114                 0.209                  0.424
## 4       23    200   900 -10205                 0.140                  0.5
## 5       23    200   700 -10207                 0.140                  0.357
## 6       19    200   900 -10326                 0.209                  0.568
## # ... with 4 more variables: right_guess_percentage <dbl>, reward <dbl>,
## #   penalty <dbl>, timeframe <dbl>
```

Variables' units:

- `duration`, `timeframe`: second
- `radius`, `range`: millimeter

For now, the scoring system seems to be discouraging the formulation of `stops`. It appears that it would rather not to have `stops` than to have too many by raising the value of `duration` and lowering `radius`. Better combination of `reward` and `penalty`values are of utter necessity.

Percentage System

While the scoring system conduct scoring with `reward` and `penalty` repeatedly, the percentage system only counts of the unique events during a `timeframe` to yield five percentages, ranging from 0 to 1, as indicator of alignment between position and observation data.

The first three percentages: `percentage of stops in position only`, `percentage of stops in observation only`, `percentage of stops in both`, pertains to the way we data mine `stops`, i.e., the `duration` and `radius` parameters. The algorithm is rather straight forward. By going through the observation and position data, where stops are data mined with given `duration` and `radius` parameters, the program count the number of `stops` that only appears in position or only in observation. The count of mutually admitted `stops` is calculated as the half of the total number of `stops` minus the aforementioned two counts. Below are the parameter combinations that performs the best with `percentage of stops in both`:

```
## # A tibble: 6 x 8
##   duration radius perc_of_stops_in_both perc_of_stops_i~ perc_of_stops_i~ reward
##      <dbl>  <dbl>                 <dbl>            <dbl>            <dbl>  <dbl>
## 1       23   1800                 0.755            0.153           0.0918      2
## 2       19   1800                 0.744            0.190           0.0662      2
## 3       19   1600                 0.742            0.171           0.0869      2
## 4       15   1800                 0.741            0.230           0.0287      2
## 5       23   1600                 0.737            0.132           0.132       2
## 6       27   1800                 0.729            0.130           0.142       2
## # ... with 2 more variables: penalty <dbl>, timeframe <dbl>
```

As the above three percentages take care of the `duration` and `radius` parameters, the `right guess percentage` and `right match percentage` measures how `range` works together with `duration` and `radius` by comparing the contents in `potential subject sets`. As mentioned before, a `timeframe` is established a stopping event in the observation data is encountered. We then take the **unique** `potential subject sets` from this `timeframe` in the position data. Each **unique** set constitutes a `match` attempt while each element in these sets constitutes a `guess` attempt. A correct `match` or `guess` is found when an element in the `potential subject set` is identical with the true subject denoted in the observation data.

We want to differentiate between `match` and `guess` since the former encourages larger `range`, because it does not hurt for `match` just to throw a large amount of element into `potential subject set` as the total number of `matches` will not change. We also do not want to just include `guess` since the algorithm will prefer a small value of `range` and have `potential subject set` almost always empty. By combining both, we wish to find a compromised solution in the future.

Below are the parameter combinations with the best `match` rate:

```
## # A tibble: 6 x 10
##   duration radius range  score perc_of_stops_in_both right_match_percentage
##      <dbl>  <dbl> <dbl>  <dbl>                 <dbl>                   <dbl>
## 1       19    200  1300 -12673                 0.209                   0.688
## 2       27    200  1300 -11338                 0.147                   0.648
## 3       23    200  1300 -11669                 0.140                   0.643
## 4       15    200  1300 -13980                 0.272                   0.639
## 5       11    200  1300 -16090                 0.390                   0.632
## 6        7    200  1300 -19057                 0.429                   0.631
## # ... with 4 more variables: right_guess_percentage <dbl>, reward <dbl>,
## #   penalty <dbl>, timeframe <dbl>
```

Below are the parameter combinations with the best `guess` rate:

```
## # A tibble: 6 x 10
##   duration radius range  score perc_of_stops_in_both right_match_percentage
##      <dbl>  <dbl> <dbl>  <dbl>                 <dbl>                   <dbl>
```

```
## 1       11    200   100 -14639                0.390                0.0120
## 2        7    600   100 -24506                0.525                0.00215
## 3        7   1200   100 -26799                0.623                0.00210
## 4       11    600   100 -22341                0.556                0.00246
## 5       11    800   100 -24122                0.589                0.00233
## 6       15    200   100 -12988                0.272                0.0118
## # ... with 4 more variables: right_guess_percentage <dbl>, reward <dbl>,
## #   penalty <dbl>, timeframe <dbl>
```

Appendices

Appendix I: Classroom Objects Coordinates

```
## # A tibble: 39 x 3
##    object     X     Y
##    <chr>  <dbl> <dbl>
##  1 seat1   2478  4280
##  2 seat2   3327  4178
##  3 seat3    288  3702
##  4 seat4   1424  3554
##  5 seat5    519  2934
##  6 seat6   1050  2988
##  7 seat7   2498  2963
##  8 seat8   3017  2959
##  9 seat9   4026  4000
## 10 seat10  5221  3824
## # ... with 29 more rows
```

Appendix II: Stop Detection Source Code

```
############################################################
# Created by Tianze (Steven) Shou for RETTL Summer 2022
# This file serves as a utility package to detect teacher's stopping behavior
# defined in Martinez-Maldonado R, et al.
# Full citation below:
# Roberto Martínez-Maldonado, Lixiang Yan, Joanne Deppeler, Michael Phillips, Dragan Gašević, Classroom
############################################################


import math

def getObsStopEvents():

    """
    :return: returns a set of strings, which are all stopping event names in distilled observation log
    """

    return {"Talking to student: ON-task",
            "Talking to student: OFF-task",
            "Talking to small group: ON-task",
            "Talking to small group: OFF-task"}

def cols2tuples(Xcol, Ycol):
```

```
    '''
    Converts the columns representing a position in a dataframe to an array of
    tuples of the same length
    '''

    # input check
    assert len(Xcol) == len(Ycol), "Error: Xcol and Ycol of different size"

    tuplePoints = [] # initialize an output array
    for i in range(len(Xcol)):
        tuplePoints.append( (Xcol.iloc[i], Ycol.iloc[i]) )

    # return check
    assert len(tuplePoints) == len(Xcol), "Error output number of point different from input"

    return tuplePoints



def getDist(point0, point1):

    '''
    Points are represented by tuples in this funciton
    Returns the distance between point1 and point2
    '''

    X0 = point0[0]
    Y0 = point0[1]
    X1 = point1[0]
    Y1 = point1[1]

    return math.sqrt( (X0 - X1)**2 + (Y0 - Y1)**2 )


def getCentroid(points):
    sumX, sumY = 0, 0
    for point in points:
        sumX += point[0]
        sumY += point[1]
    centroid = (sumX / len(points), sumY / len(points))
    return centroid


def withinRadius(points, radius):

    '''
    @param points: an array of points (represented by tuples).
                Example: [(point1X, point1Y), (point2X, point2Y), ...]
    @param radius: radius parameter that specifies a stop, unit is milimeter
    @return: return a boolean logical of whether all points are with the radius
            range of these points' centroid
    '''
```

```
    # calculate the centroid of these points
    sumX, sumY = 0, 0
    for point in points:
        sumX += point[0]
        sumY += point[1]
    centroid = (sumX / len(points), sumY / len(points))

    for point in points:
        # return False if one point is not within radius range
        if(getDist(point, centroid) > radius): return False

    return True


def validateStops(stops, duration, maxDuration, epsilon=0.4):

    '''
    @param stops: a list of start timestamp and end timestamp of stops.
                Example: [(start0, end0), (start1, end1), ...]
    @param duration: the minimum duration that the teacher has to stay in-place
                     in order to define a stop
    @param epsilon: allows a margin for mistake in timestamp
    @return: returns a logical, indicating whether the sequence of start and stop
            times are valid
    '''

    for i in range(len(stops)-1):
        currStopStart = stops[i][0]
        currStopEnd = stops[i][1]
        nextStopStart = stops[i+1][0]
        nextStopEnd = stops[i+1][1]

        # the stop duration much reach indicated parameter
        if(currStopEnd - currStopStart < duration - epsilon or
           currStopEnd - currStopStart > maxDuration):
            print("stop", i, "is invalid")
            print("start is", currStopStart)
            print("end is", currStopEnd)
            return False
        if(nextStopEnd - nextStopStart < duration - epsilon):
            print("stop", i + 1, "is invalid")
            print("start is", nextStopStart)
            print("end is", nextStopEnd)
            return False

        # the end of current stop must be ealier than next stop's start
        if(currStopEnd > nextStopStart):
            print("stop", i, "and", i+1, "are overlapping")
            return False

    return True
```

```
def getStops(X, Y, timestamp, periods, days, duration, radius):

    '''
    @param X: an numpy array of x-coordinates
    @param Y: an numpy array of y-coordinates, must be same length as X
    @param timestamp: an array of timestamps, must be same length as X and Y
    @param duration: the minimum duration that the teacher stays in-place to
                    establish a stop. Unit is second
    @param radius: teacher position must be with in the radius of coordinate
                centroid to establish a stop. Unit is milimeter
    @return: an array of tuple (<stopStartTime>, <stopEndTime>)
    '''

    assert len(X) == len(Y), "Lengths of X, Y coordinate arrays should be identical"
    assert len(timestamp) == len(X), "Lengths timestamp array should be identical to coordinate arrays"

    maxDuration = 600 # a stop should not exceed 10 minutes

    # arrange X, Y coordinates into a array of tuples
    pos = []
    for i in range(len(X)):
        pos.append( (X[i], Y[i]) )
    assert(len(pos) == len(timestamp))

    stops = [] # output array
    # format:[ (<stop_start_time>, <stop_end_time>), ... ]

    # loop thru the pos array in the time frame specified by `duration`
    startIndex = 0
    while(startIndex < len(pos) - duration):
        endIndex = startIndex + duration
        # look further down position points if all points are within radius
        while(withinRadius(pos[startIndex: endIndex], radius) and
              endIndex < len(pos) and
              # the following condition is to ensure that we do not detect
              # a stop that goes into two periods or days
              periods[startIndex] == periods[endIndex] and
              days[startIndex] == days[endIndex]):
            endIndex += 1

        # edge case where endIndex advances to the next day/period
        # edge case where enIndex is out of bound by 1
        if(endIndex >= len(pos) or
           periods[startIndex] != periods[endIndex] or
           days[startIndex] != days[endIndex]):
            endIndex -= 1

        # this means that no stop is detected
        if(endIndex <= startIndex + duration):
            # move the timeframe to the next second
            startIndex += 1
        # a stop is detected
        else:
```

```
            # pass this stop to output array
            if(endIndex >= len(pos)): endIndex = len(pos) - 1 # to catch the edge case
            stops.append( (timestamp[startIndex], timestamp[endIndex]) )
            # start of next timeframe should be the end of this stop
            startIndex = endIndex

    assert(validateStops(stops, duration, maxDuration))
    return stops

def getStopsFromObs(obsLog):

    """
    :param obsLog: distilled observation data file
    :return: returns a list of timestamps signaling the start of each stopping events in observation lo
    """

    stops = [] # to be returned
    # go through observation log to find the stopping events
    for i in range(len(obsLog)):
        if( obsLog.iloc[i]["event"] in getObsStopEvents() ):
            stops.append(obsLog.iloc[i]["timestamp"]) # attach the stopping event's timestamp to the li

    return stops
```

References

[1] Roberto Martínez-Maldonado, Lixiang Yan, Joanne Deppeler, Michael Phillips, Dragan Gašević, Classroom Analytics: Telling Stories About Learning Spaces Using Sensor Data, Hybrid Learning Spaces, 10.1007/978-3-030-88520-5_11, (185-203), (2022).