Master-Praktikum: IoT (Internet of Things)
Sensor Node Development

Ozmen, Atalay Sirikci, Hande

July 8, 2023

1 Problem Definition

Our goal in this part is to design and implement an algorithm on microcontroller ESP32 to count the number of people in seminar room by receiving the photoelectric barrier signals.

2 Our Approach

Rather than handling the signals coming from barriers separately, we decided to store states that include information of both barriers. This will help to detect the actions and also peek cases. Since there are two barriers on the door, four possible states are represented as below:

$\mathrm{b}_{\mathrm{in}} b_{\mathrm{out}}$	
0 0	No activated barrier
0 1	Only outer barrier broken
1 0	Only inside barrier broken
1 1	Both barriers activated

Table 1: States of barriers

- Section 2.1 Logic of the algorithm
- Section 2.2 Implementation details
- Section 2.3 Debouncing strategy
- Section 2.4 How we handle corner cases

2.1 Finite State Machine

Since particular inputs can cause particular changes in the program state, we preferred to use finite state machine. Inputs of the system is the state of the barriers.

$$\begin{tikzcd}
 & B \arrow[rr, bend left, "\text{(I) 10}"] \arrow[dl, "11"] \arrow[dr, bend left, "00"] & & E \arrow[ll, "\text{(II) 01}"] \arrow[dl, "00"] \arrow[dr, bend left, "11"] & \\
C \arrow[ur, bend left, "01"] \arrow[rr, bend right, "00"] \arrow[dr, "10"] & \text{start} \arrow[r] & A \arrow[ul, "01"] \arrow[ur, bend left, "10"] & & F \arrow[ul, "10"] \arrow[ll, "00"] \arrow[dl, bend left, "01"] \\
 & D \arrow[ul, bend left, "11"] \arrow[ur, bend right, "\text{00, counter++}"] & & G \arrow[ul, "\text{00, counter–}"] \arrow[ur, "11"] &
\end{tikzcd}$$

Whenever program starts, we start from A which is the initial state. During the execution, we jump to A whenever we receive 00 as the input.
Arrow labeled as (I) refers to the case where we receive barrier status 10 just after 01, and arrow labeled as (II) is receiving barrier status 01 just after 10. Those are the cases we don't expect to face but handled them by continuing from the very last state that we receive.

A valid path to increase the counter is:

$$\begin{tikzcd}
\text{start} \arrow[r] & A \arrow[r, bend left, "01"] & B \arrow[l, "\text{(I) 00}"] \arrow[r, bend left, "11"] & C \arrow[l, "\text{(II) 01}"] \arrow[r, bend left, "10"] & D \arrow[l, "\text{(III) 11}"] \arrow[lll, bend left, "\text{00, counter++}"]
\end{tikzcd}$$

A valid path to decrease the counter is:

$$\begin{tikzcd}
\text{start} \arrow[r] & A \arrow[r, bend left, "10"] & E \arrow[l, "\text{(I) 00}"] \arrow[r, bend left, "11"] & F \arrow[l, "\text{(II) 10}"] \arrow[r, bend left, "01"] & G \arrow[l, "\text{(III) 11}"] \arrow[lll, bend left, "\text{00, counter–}"]
\end{tikzcd}$$

Backward arrows (I, II, III) enable algorithm to handle ”Unsure to enter/leave Room” cases. By this way; no matter the person attempts to enter/leave but steps back in any stage, algorithm will start over from the very last position and keeps the history.

2.2 Internal Structure of the Code

2.2.1 Tasks

Our implementation is mainly consisting of tasks, which we create in `app_main` and assign priorities to tasks. Here are the priorities of the tasks:

Task	Priority	Reason
`vUpdatePrediction`	20	runs every 5 minutes and we need to update it as soon as possible
`vReceivingSignal`	15	receiving data should be prioritized to make the system accurate
`vProcessingSignal`	14	processing can wait since we prefer accuracy over low latency
`textDemo`	13	display can wait since latency is not critical
`vSendMessages`	10	sending messages to Kibana can wait, it slightly effects the prediction model

Table 2: Tasks

2.2.2 Data Structures and Code Flow

variable	
`event`	object where we store the barrier status and timestamp
`xDetectionQueue`	queue where we store the signals to process
`counter`	holds the number of people in seminar room
`prediction`	holds the prediction value
`lastFSMState`	last state of FSM, initial value is 0 which refers to A
`lastBarrierState`	last barrier status, for debouncing strategy

Table 3: Tasks

We first receive the signals by the `vReceivingSignal` and it adds the object of `event` into `xDetectionQueue` after the necessary checks. `vProcessingSignal` gets the `event` objects from the queue and calls the `stateTransition` function to move on FSM.

2.2.3 Interrupt

Rather than constantly checking the change of status in barriers, we preferred to use interrupt that get notified whenever any change occurs. In function named `gpio_isr_handler` we bind the functions should be triggered by the interrupts.

2.3 Debouncing Strategy

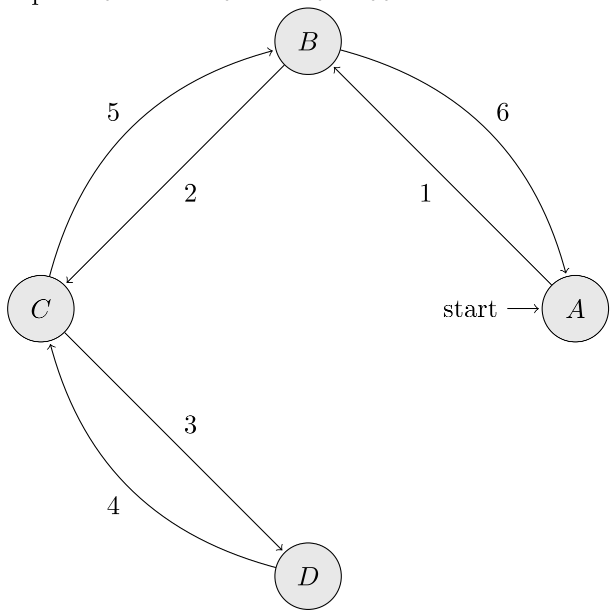
To solve the debouncing problem, we decided to keep the last barrier state `lastBarrierState` and check whether the new input barrier state is not equal to `lastBarrierState` in order to put the input into `xDetectionQueue`. By this way, we ignore the repeating barrier states.

Including the state of each barrier in `lastBarrierState` and `event` object helped us to deal with debouncing in easier way since removing repeating states of just one barrier would cause different results than removing repeating states of both barriers.

2.4 Corner Cases

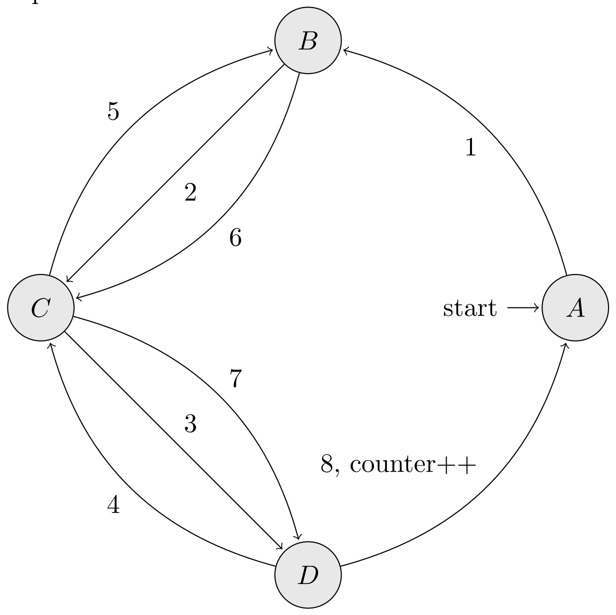
peekIntoBothRoom:

Inputs: 01 - 11 - 10 - 11 - 01 - 00



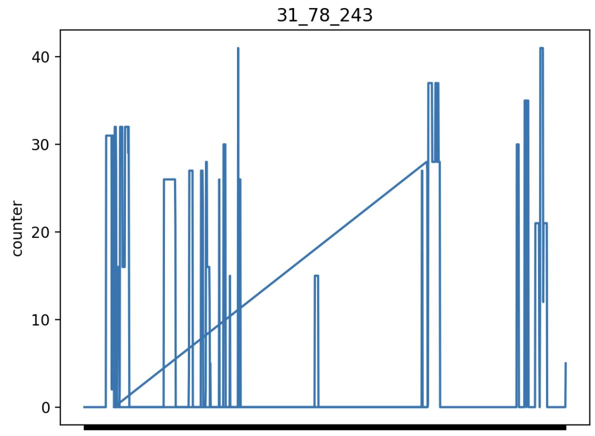
unsureEnter:

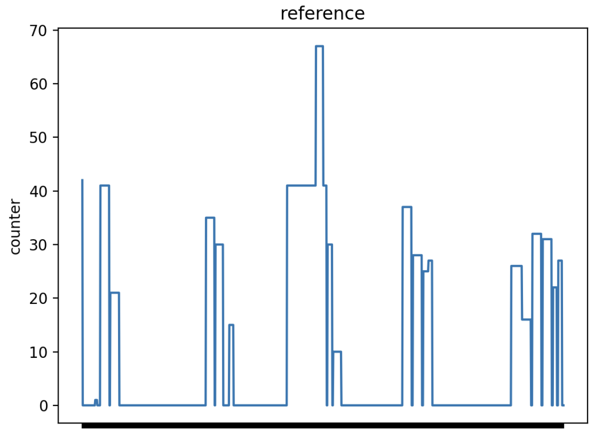
Inputs: 01 - 11 - 10 - 11 - 01 - 11 - 10 - 00

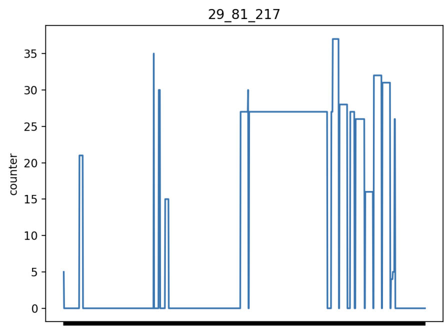


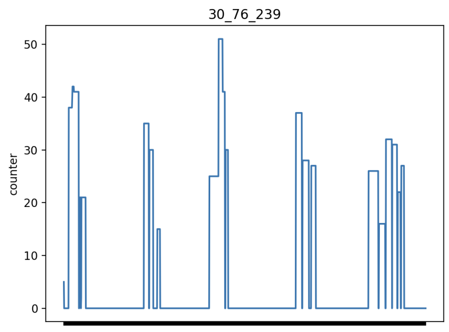
3 Our Data

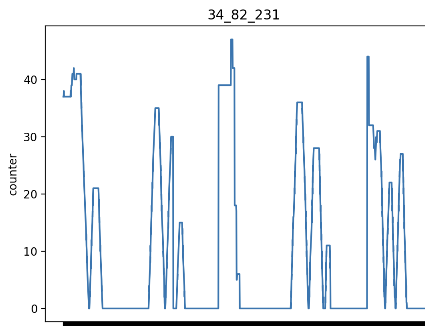
Below you can see our data (index: 31_78_243), reference data and other groups' data values for the time duration of Jun 26, 2023 00:00:00.000 - Jun 30, 2023 23:59:59.999

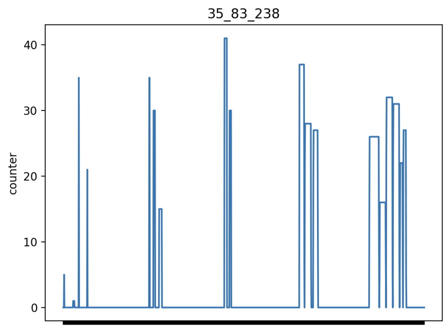

31_78_243
counter

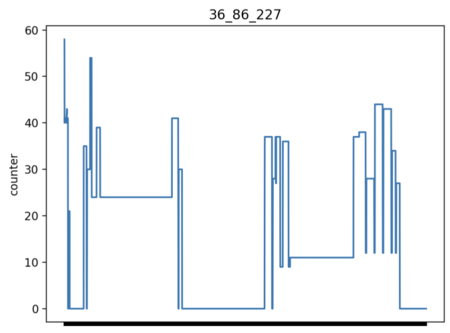

reference
counter

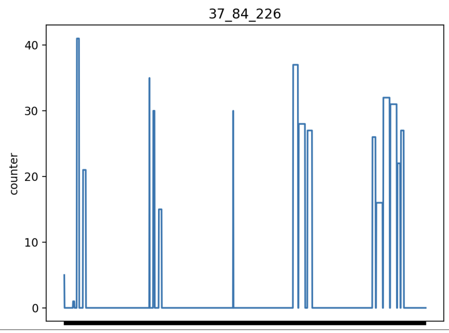

29_81_217
counter

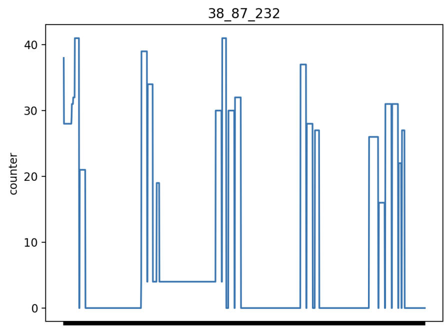

30_76_239
counter

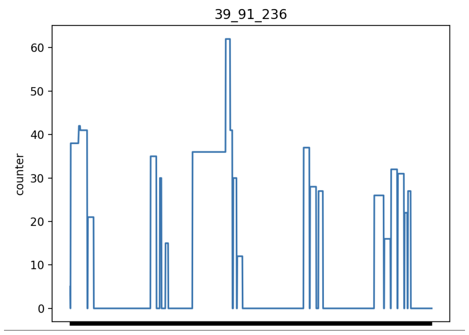

34_82_231
counter

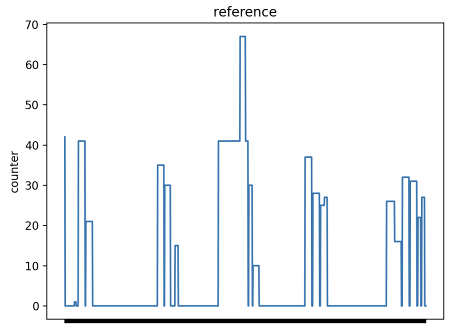

35_83_238
counter


36_86_227
counter


37_84_226
counter


38_87_232
counter


39_91_236
counter


reference
counter