Please adjust group name in tumuser.sty.
Department of Electrical and Computer Engineering
Technical University of Munich

Counting Algorithm

Group 09

Matthias Hainz ✉ **and Alexandra Seibicke** ✉

Department of Electrical and Computer Engineering, Technical University of Munich

✉ matthias.hainz@tum.de
✉ alexandra.seibicke@tum.de

July 15, 2022

Abstract — This short paper summarizes the counting algorithm implemented on Group 09's ESP32 device as part of the task for the Master-Praktikum: IoT.

1 Introduction

This article is about the code which was written and deployed to our ESP32 device for the purpose of counting the students, which are entering and leaving the Seminarroom 5606.01.020 and thus keeping track of the current number of people in the room. For this purpose, the ESP32 board is connected to two light barriers, which are placed at the inner and the outer side of the door respectively. Those barriers are connected to the pins 18 (outer) and 19 (inner). We'd like to take a look at the implemented counting algorithm, the mehtod used to handle signal bouncing and the corner cases it handles.

1.1 The Counting Task

The counting algorithm is implemented in a seperate task and is thus running in an indefinite loop. At device startup, the pins 18 and 19 are set two function as input pins, to enable signal receiving from these two pins. Afterwards we enabled s software pulldown resistor, to convert the analog signal to a digital 0 value. At the startup of the Task, one interrupt handler is assigned to each of the two pins and therefor for each of the two light barriers. With those set, we were able two receive any change in signal sent from the barriers. We chose the detection of ANYEDGE, meaning a signal change from 0 to 1 as well as from 1 to 0, as we use both, the breaking and the reestablishing of the light barriers in our counting algorithm. For each of the handlers, we wrote an according method to process the change in the signal.

1.2 The State Machine

To support our algorithm, we decided to use a state machine in order to keep track of which barriers where breached and reestablished in which order. Figure 1 shows the possible transitions and how they are achieved. This state machine can keep track of every change in position a convex body can make while it is in the reach of the light barriers. By keeping track of which barrier was breached first, we can determine wheter a person moves out of or into the room and thus increase or decrease the count accordingly. This change in count occures, when the IDLE state is reached from the SECONDONLY state, which indicates, that a person moved through the door.

2 The Algorithm

Based on this state Machine, we implemented an algorithm to keep track of the state in which we currently are. The default state is always IDLE and thus by breaching any of the two barriers, we always transition to the FIRSTONLY state. Based on which barrier actually was breached, we then set the boolean variable 'innerFirst' to true or false. 'innerFirst' = true indicates that the person who entered the door is currently leaving the room, while 'innerFirst' = false indicates the contrary. Thus, this variable shows, which barrier is the 'first' and which the 'second' referred to in Figure 1. The last thing we need to know to finally determine which state transition to take is, if the barrier triggering the interrupt was breached or reestablished. As we had problems with measuring the current signal correctly due to the bouncing occuring after a signal was retrieved, we decided to keep track of it manually, by flipping a bit every time we retrieve an interrupt. Based on the current state, the barrier, which triggered an interrupt, its new digital signal and the 'innerFirst' variable we can does determine, how to proceed in our state machine. For further unterstanding, i will provide an example of a scenario: We receive an interrupt from the outer barrier, proceed to flip the state-bit for this barrier and determine that the barrier was just reestablished. This means, that we cannot be in the IDLE state, as then both barriers would already have

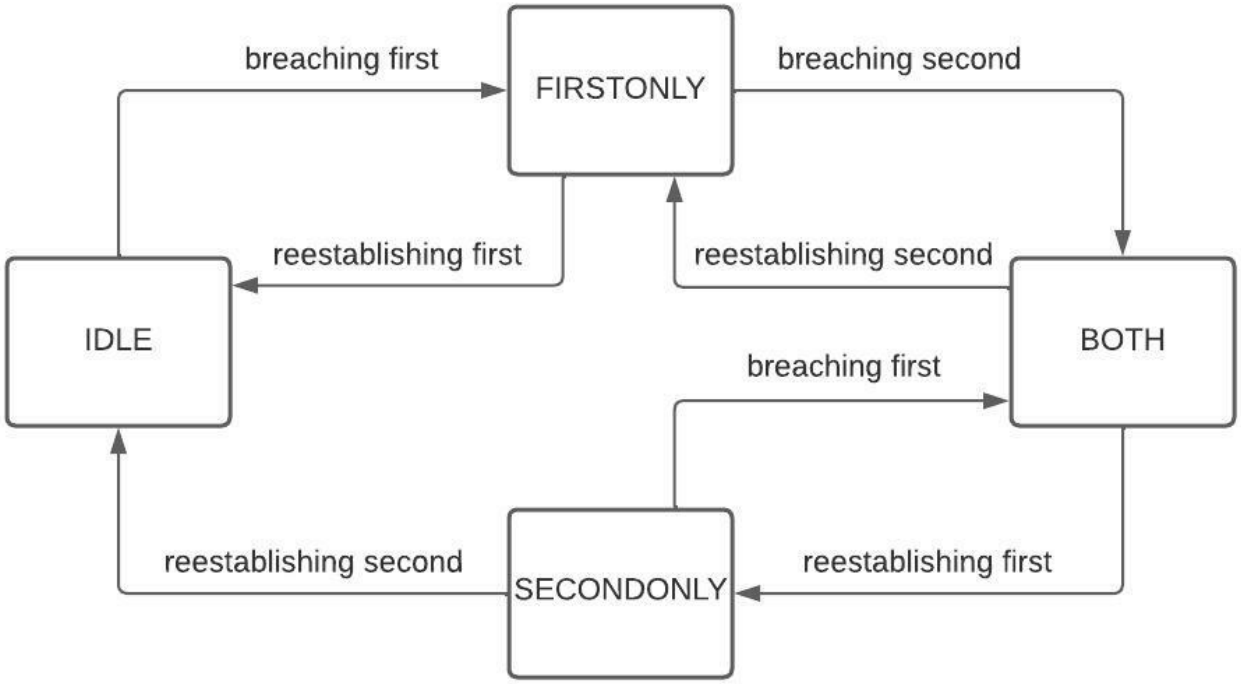

Figure 1 'first' and 'second' refer to the light barriers which are breached first or second in one iteration of the state machine befor reentering the IDLE state

been established before. Next we check, if 'innerFirst' is true. We see that this is indeed the case, and that we thus have one of the 'reestablishing second' transitions, which only start from the BOTH or the SECONDONLY states. Last we thus check for the current state and find it being SECONDONLY. Therefor we proceed to the IDLE state and decrease the count, as 'innerFirst' = true indicates that a person is currently leaving the room.

3 Signal Debouncing

Whenever a signal change was retrieved from one of the barriers, a bouncing could be measured in the signal, before it went back to its current digital value again. This bouncing led to the detection of a high number of interrupts with each interaction with one of the barriers, as each of those bounces was recognized as a change of signal. This naturally caused a weird behavior in the state machine, as each of the detected interrupts would be processed and could possibly lead to a changed state. Tp battle this issue, we implemented a time check, which assures that only one signal is handled from each barrier every so and so often. After experimenting with the timespan, we realized that bouncing mostly occurs in the first two milliseconds after the actual signal. To be sure, we thus set the timespan in which no other signals are processed to 5 milliseconds. The code in Listing 1 shows the if statement executed before further signal processing for the inner barrier. 'milliseconds' describes the current number of milliseconds since the UNIX Epoch, while 'lastinnermilliseconds' describes the number of milliseconds since the UNIX Epoch, when the last inner barrier signal was handled. 'debounce time' stores the number of milliseconds which we wait before accepting new signals to enable debouncing. The same is statement is executed when handling the outer barrier signals, but there the last processing timestamp is stored on the 'lastOuterMilliseconds' variable.

```
1 if(milliseconds > lastInnerMilliseconds +
      debounceTime){
2   ...
3 }
```

Listing 1 debounce handling

4 Corner Cases

Due to the complete tracking of every light barrier signal and the full state machine, which is capable of transitioning back and forwards, our code can detect every possible corner case, when people are viewed as convex shapes moving in and out through the door and only one person is allowed in the doorframe at

a time. These corner cases include the peaking into/out of the room followed by backing out/in again, as well as students walking almost entirely through the door and then turning on the heel and deciding to go back. Wiggling back and forth inside of the doorframe can also be handled as the count is only decreased/increased, when reentering the IDLE state and when the person left the doorframe on the other side that it entered it. However this works well in theory and in a controlled environment the corner case detection has some drawbacks. Still regarding an isolated person, which moves through the door with swinging arms or a backpack with something dangling down behind the person, the state transitions might lead to a weird end state after the person passed the door. The conducted tests have shown that the passing of a person was recognized correctly in almost all cases, but the swinging arms led to a change of state after the person has passed. The next person than would enter the door, while we are not in the IDLE state which led to a weird and unpredictable behaviour of the algorithm. Therefor, we added another check before the signal handling to reset the state to IDLE if the last signal handling occured more than a second ago. Listing 2 shows the full actions which are taken before the signal is processed. After the debouncing, we check if one second passed since any signal was handled the last time. The milliseconds timestamp for this is stored in 'lastMilliseconds'. If the last handling indeed occured more than a second ago, we reset the state to IDLE and set the bits for the inner and outer barrier to 0. innerState = false and outerState = false indicate that the respective barriers are currently established. After that we proceed with flipping the bit for the inner barrier, as we are about to handle a interrupt triggered by the inner barrier and we set the 'lastMilliseconds' and the 'lastInnerMilliseconds' to the current number of milliseconds since the UNIX Epoch which is still stored in the 'milliseconds' variable.

```
if(milliseconds > lastInnerMilliseconds +
    debounceTime){
  if (milliseconds > lastMilliseconds +
    1000){
    state = IDLE;
    innerState = false;
    outerState = false;
  }
  innerState = !innerState;
  lastMilliseconds = milliseconds;
  lastInnerMilliseconds = milliseconds;
    ...
}
```

Listing 2 This code shows the actions taken before the actual processing of the retrieved signal

5 Drawbacks

One Drawback of the full state machine approach we chose to enable the handling of all different kinds of corner Case handling is, that two persons passing through the door closely after each other are more likely to be detected as only one person. When using an approach which does allow less corner case detection, the rate of correctly detected student pairs entering the room can be decreased. Taking a look at the state transitions which are visualized in Figure 2, we can see, that an implementation of this algorithm would only care for people, stepping far enough into the door to count them as passing through the door.

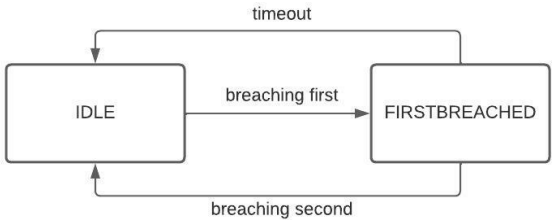

Figure 2 'first' and 'second' refer to the light barriers which are breached first or second

Applying both algorithms to the scenario described in Figure 3, we would get following state tranisitions in one instance of our state machine: IDLE -> FIRSTONLY -> BOTH -> SECONDONLY -> BOTH -> FIRSTONLY -> BOTH -> SECONDONLY -> IDLE resulting in the detection of one person passing through the door.

For the state machine in Figure 2, two instances would be created and only Figure 3.1, 3.2, 3.4 and 3.6 would be interesting for the transitions, leading to the IDLE -> FIRSTBREACHED -> IDLE in both instances of the state machine and detecting both passing people. This method however would be prone to errors which are caused by e.g. swinging arms. We thus decided to use the method we continued to implement and was described above.

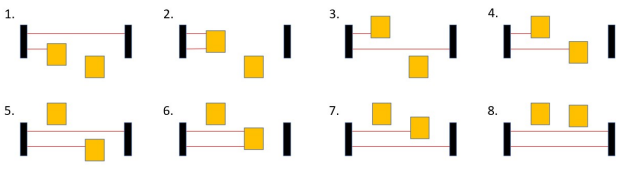

Figure 3 'Two objects passing the light barriers close to each other