Technical University of Munich

Informatics 10 – Chair of Computer Architecture and Parallel Systems (Prof. Schulz)

Master Praktikum: IoT (Internet of Things) (IN2106, IN4224)

MILESTONE 1, counting PDF report

Group: 2

Students and matriculation number:

Nadija Borovina (03742897)

Nedžad Hadžiosmanović (03742896)

Lecturers:

Gerndt, Hans Michael, Prof. Dr.

Podolski Vladimir, M. Sc.

Chenyuan Zhao

Munich, May 2021.

For the purpose of explaining the counting mechanism that our group implemented we made a state diagram, using UML Star, because we implemented the mechanism of counting as a finite state machine. Before explaining the state diagram and its components, all the elements which can be seen in the diagram are going to be introduced, both the ones which will represent the states, and the ones which will be events that will trigger a change between states.

According to our implementation, the embedded system (which we will refer to as "machine" in the following text) can be in one of 11 states:

1. startState
2. enteringOutsideActiveInsideInactive
3. enteringBothActive
4. enteringBothInactive
5. enteringOutsideInactiveInsideActive
6. enteringStateFinishBothInactive
7. exitingOutsideInactiveInsideActive
8. exitingBothActive
9. exitingBothInactive
10. exitingOutsideActiveInsideInactive
11. exitingStateFinishBothInactive

The changes in the state of machine depend on timers and actions (and constants) that we defined.
The timers that influence the change of state of the machine used are:

1. timerForAvoidingObstruction
2. timerUntilEnteringOrEnteringOrExitingRoom
3. startedEntering
4. startExiting
5. middleStateTimer

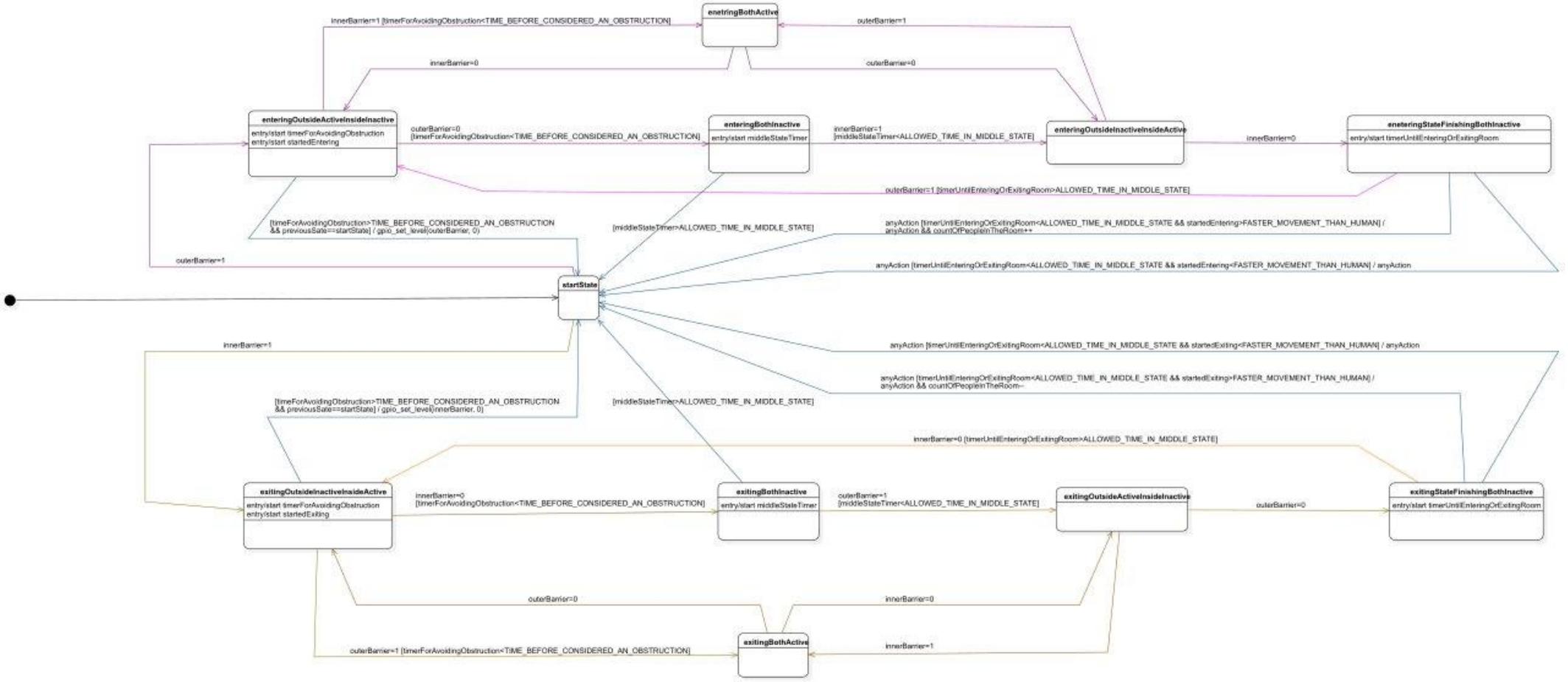
The actions which influence the change of state of the machine used in our implementation are:

1. risingEdgeOutside
2. fallingEgdeOutside
3. risingEdgeInside
4. fallingEdgeInside

The constants, which in combination with timers and action influence the change of states of the machine used in the implementation of our group are:

1. ALLOWED_TIME_IN_MIDDLE_STATE
2. TIME_BEFORE_CONSIDERED_AN_OBSTRUCTION
3. FASTER_MOVEMENT_THAN_HUMAN

For a better understanding how does our mechanism of counting work, on the next page you can see the state diagram that my group made for the embedded system. (Note: For a better resolution photo than the one provided below, you can visit the following link, on which you will find the same diagram as a file with "svg" extension:
https://drive.google.com/drive/folders/1ISETgGBcizN-vj3F91zueaHJIOLXnBRj?usp=sharing)


eneteringBothActive
innerBarrier=1 [timerForAvoidingObstruction<TIME_BEFORE_CONSIDERED_AN_OBSTRUCTION]
outerBarrier=1
innerBarrier=0
outerBarrier=0
enteringOutsideActiveInsideInactive
entry/start timerForAvoidingObstruction
entry/start startedEntering
outerBarrier=0
[timerForAvoidingObstruction<TIME_BEFORE_CONSIDERED_AN_OBSTRUCTION]
enteringBothInactive
entry/start middleStateTimer
innerBarrier=1
[middleStateTimer<ALLOWED_TIME_IN_MIDDLE_STATE]
enteringOutsideInactiveInsideActive
innerBarrier=0
eneteringStateFinishingBothInactive
entry/start timerUntilEnteringOrExitingRoom
outerBarrier=1 [timerUntilEnteringOrExitingRoom>ALLOWED_TIME_IN_MIDDLE_STATE]
[timeForAvoidingObstruction>TIME_BEFORE_CONSIDERED_AN_OBSTRUCTION
&& previousSate==startState] / gpio_set_level(outerBarrier, 0)
[middleStateTimer>ALLOWED_TIME_IN_MIDDLE_STATE]
anyAction [timerUntilEnteringOrExitingRoom<ALLOWED_TIME_IN_MIDDLE_STATE && startedEntering>FASTER_MOVEMENT_THAN_HUMAN] /
anyAction && countOfPeopleInTheRoom++
outerBarrier=1
anyAction [timerUntilEnteringOrExitingRoom<ALLOWED_TIME_IN_MIDDLE_STATE && startedEntering<FASTER_MOVEMENT_THAN_HUMAN] / anyAction
startState
innerBarrier=1
anyAction [timerUntilEnteringOrExitingRoom<ALLOWED_TIME_IN_MIDDLE_STATE && startedExiting<FASTER_MOVEMENT_THAN_HUMAN] / anyAction
anyAction [timerUntilEnteringOrExitingRoom<ALLOWED_TIME_IN_MIDDLE_STATE && startedExiting>FASTER_MOVEMENT_THAN_HUMAN] /
anyAction && countOfPeopleInTheRoom--
[timeForAvoidingObstruction>TIME_BEFORE_CONSIDERED_AN_OBSTRUCTION
&& previousSate==startState] / gpio_set_level(innerBarrier, 0)
[middleStateTimer>ALLOWED_TIME_IN_MIDDLE_STATE]
innerBarrier=0 [timerUntilEnteringOrExitingRoom>ALLOWED_TIME_IN_MIDDLE_STATE]
exitingOutsideInactiveInsideActive
entry/start timerForAvoidingObstruction
entry/start startedExiting
innerBarrier=0
[timerForAvoidingObstruction<TIME_BEFORE_CONSIDERED_AN_OBSTRUCTION]
exitingBothInactive
entry/start middleStateTimer
outerBarrier=1
[middleStateTimer<ALLOWED_TIME_IN_MIDDLE_STATE]
exitingOutsideActiveInsideInactive
outerBarrier=0
exitingStateFinishingBothInactive
entry/start timerUntilEnteringOrExitingRoom
outerBarrier=0
innerBarrier=0
exitingBothActive
outerBarrier=1 [timerForAvoidingObstruction<TIME_BEFORE_CONSIDERED_AN_OBSTRUCTION]
innerBarrier=1

First thing that should be noted about the diagram itself is that it can be separated in two part which are very similar, and that the first state that the machine is in when it is turned on is the "*startState*".
The first part of the state diagram is part of the diagram that represents states in which the machine can be because a person is entering the room. The part of the diagram representing this is the **upper part of the diagram**, where the lines that represent transitions from one state to another are colored in purple (if they are going from a state that belongs to the upper part of the diagram to another state that belongs to the upper part of the diagram), and blue (if they are going from a state that belongs from the upper part of the diagram to the "*startState*").
Opposite, the second part of the state diagram is the one that represents states in which the machine can be because a person is exiting the room, which is the **lower part of the diagram**. In this part of the diagram the lines that represent transitions from one state to another are colored in orange (if they are going from a state that belongs to the lower part of the diagram to another state that belongs to the lower part of the diagram) and blue (if they are going from a state that belongs from the lower part of the diagram to the "*startState*").

Let's start with examining the upper part of the state diagram. In order to start entering the room, someone from the outside would need to break the outer light barrier (causing the outer light barrier to go from logical 0, to a logical 1), which would cause an event called "*risingEdgeOutside*". Upon detecting this event, the machine changes its state from "*startState*" to "***enteringOutsideActiveInsideInactive***". When the machine is entering this state, it starts two timers, "*timerForAvoidingObstruction*" and "*srartedEntering*".
The timer "***timerForAvoidingObstruction***" is a timer which will wait for a certain number of millisecond (which is defined by the constant "*TIME_BEFORE_CONSIDERED_AN_OBSTRUCTION*") before considering the object that broke the light barrier as an obstruction. If someone or something was standing in the light barrier for some time, without moving, it would prevent the machine from being able to register people coming in and out of the room, which is its main purpose. An example how this could happen is if somebody leaned on one side of the door frame and stayed there for some time, or that someone moved some object in a part of the door in which the barrier detects the object. In any of the cases, if it is a person or an object that is standing in the light barrier for "*TIME_BEFORE_CONSIDERED_AN_OBSTRUCTION*", the machine will detect it as an obstruction and change the state of the machine back to the "*startSate*".
The second timer started in the "*enteringOutsideActiveInsideInactive*" called "***srartedEntering***" is just counting time from when "*enteringOutsideActiveInsideInactive*" is entered, so that it can be compared later on to the time passed after the machine goes through sequences from the upper part of the state diagram and finally reaches the "*enteringStateFinishBothInactive*". The comparison between time passed from when the timer was started is used to see if the change of machine states between these two states happened too quickly, in a sense that a human being can go from "*enteringOutsideActiveInsideInactive*" to "*enteringStateFinishBothInactive*" in best case scenario (or said in other words, the fastest person in the world) in a number of milliseconds defined with constant "*FASTER_MOVEMENT_THAN_HUMAN*". If the machine came from "*enteringOutsideActiveInsideInactive*" to "*enteringStateFinishBothInactive*" in less than "*FASTER_MOVEMENT_THAN_HUMAN*", this would mean that it could not be done by a human, but rather something that moves much faster, like a bullet from a gun, because of which the machine would go from "*enteringStateFinishBothInactive*" to "*startState*" without increasing the count of the people in the room.

Let's now move on and examine the "***enteringBothInactive***" state. This is a state in which the machine can come to only if it is coming from the previously explained state "*enteringOutsideActiveInsideInactive*", when event "*fallingEgdeOutside*" is detected. As soon as we enter this state a timer called "*middleStateTimer*" is started. To which state does our machine go

from this state depends on do we experience an event "*risingEdgeInside*" in less milliseconds than defined by "*ALLOWED_TIME_IN_MIDDLE_STATE*". If we don't detect that event, the machine would go back to the "*startSate*", while if we did, the machine would go to "*enteringOutsideInactiveInsideActive*". What is represented by the time defined in the constant "*ALLOWED_TIME_IN_MIDDLE_STATE*" is the most time a human being needs to consecutively 'free' the outer light barrier, and break the inner light barrier. Most simply put, as a person cannot stand between barriers (at least to our implementation, which was done assuming that the light barriers could always be put on the doorframe with a distance small enough that between them no human can fit in), there is a time in which the inner light barrier need to be broken after the outer one is 'freed'.
But if person cannot stand between the doorframe, how do we interpret that there was no action "*risingEdgeInside*" experienced then? Well, this action is guaranteed to happen if a person is entering the room, but if this action was not detected, this would mean that a person was actually not entering the room, but rather peeking from the outside, breaking the outer light barrier while peeking, and then moving away from the room, and this is exactly the reason why if "*ALLOWED_TIME_IN_MIDDLE_STATE*" passes and we don't experience "*risingEdgeInside*", the machine changes the state to "*startState*".

The state "***enteringBothActive***" is a very intuitive state, in which both of the light sensors are broken at the same time. This might happen if a person is standing in the way of a light barrier on either of the sides, and he decides to turn horizontally in the doorframe, which would lead to breaking both light barriers at the same time. Coming to this state could be either from state "*enteringOutsideActiveInsideInactive*" after detecting event "*risingEdgeInside*", or from state "***enteringOutsideInactiveInsideActive***" after detecting event "*risingEdgeOutside*", and as well we could go from this state only to the same two states, depending on the action which we detect.

The last state from the upper part of the state diagram is "***eneteringStateFinishingBothInactive***", and it is a state only reachable from the "enteringOutsideInactiveInsideActive". This is a state in which upon entering timer "*srartedEntering*" (which we already said that was set upon entering the "*enteringOutsideActiveInsideInactive*") is compared to "*FASTER_MOVEMENT_THAN_HUMAN*". These two are compared to see if the change of machine states from "*enteringOutsideActiveInsideInactive*" to "*eneteringStateFinishingBothInactive*" came too quickly to be done by a human being, and if the comparison shows that this could not be done by a human (i.e. that "*srartedEntering*" was smaller than "*FASTER_MOVEMENT_THAN_HUMAN*"), the machine changes the state to "*startState*". On the other hand, if the comparison showed this could have been done by a human being, then two things might occur, depending on the fact if we detect an activity "*risingEdgeOutside*" in a number of milliseconds defined by constant "*ALLOWED_TIME_IN_MIDDLE_STATE*" or not. If we do not experience this event in "*ALLOWED_TIME_IN_MIDDLE_STATE*", the counter simply increases, and the machine changes the state to "*startState*". But now comes the question why did we use the same constant "*ALLOWED_TIME_IN_MIDDLE_STATE*" as the time that need to pass after entering "*eneteringStateFinishingBothInactive*" in order to increase the count? Well, when we were in the "*enteringOutsideInactiveInsideActive*" state, and after we experienced event "*fallingEdgeInside*" we are not actually sure if a person went ahead and entered the room completely, or if he turned around and went the other way. In both cases, we will end up in the "*eneteringStateFinishingBothInactive*", and with help of the constant we determine which of the two scenarios happened. If the person turned around, and decided not to enter, then similarly to the situation in the "*eneterBothInactive*" state, before experiencing action "*risingEdgeOutside*" can at most pass the number of milliseconds as defined in "*ALLOWED_TIME_IN_MIDDLE_STATE*".

The logic used for the lower part of the state diagram is the same to the logic explained for the upper part, which is why we will not write about it, as by doing so we would both be repetitive, and as well we would go over the allowed wordcount.

As of the coroner cases in the file "*command_corner_cases.c*" which posted on the Moodle page of the course, all of the corner cases are covered, expect one, whit an important note that some of the times in the delays need to be converted to the constants that are above explained, to make sure that the machine works properly, without changing the logic of the corner case. So, below we are putting a table in which one column (the second one) shows original corner cases code from the provided file, while the other column (the third one) represents the changed code which is adapted for our machine. Also, in the column with the changed code, the code that has been changes it put in bold, so the reader could spot the difference between the content of the two columns easier.

	Original corner cases	Adapted corner cases
1	`/**` ` * expected count result: +1` ` */` `void enter(){` `  ESP_LOGI(TAG,"Command: Enter");` `  gpio_set_level(triggerPinOut,1);` `  vTaskDelay(100 / portTICK_PERIOD_MS);` `  gpio_set_level(triggerPinOut,0);` `  vTaskDelay(100 / portTICK_PERIOD_MS);` `  gpio_set_level(triggerPinIn,1);` `  vTaskDelay(100 / portTICK_PERIOD_MS);` `  gpio_set_level(triggerPinIn,0);` `  vTaskDelay(500 / portTICK_PERIOD_MS);` `}`	`void enter(){` `  ESP_LOGI(TAG,"Command: Enter");` `  gpio_set_level(triggerPinOut,1);` `  vTaskDelay(100 / portTICK_PERIOD_MS);` `  gpio_set_level(triggerPinOut,0);` `  vTaskDelay( **(ALLOWED_TIME_IN_MIDDLE_STATE-15)** / portTICK_PERIOD_MS);` `  gpio_set_level(triggerPinIn,1);` `  vTaskDelay(100 / portTICK_PERIOD_MS);` `  gpio_set_level(triggerPinIn,0);` `  vTaskDelay(**(ALLOWED_TIME_IN_MIDDLE_STATE+15)** / portTICK_PERIOD_MS);` `}`
2	`/**` ` * expected count result: -1` ` */` `void leave(){` `  ESP_LOGI(TAG,"Command: Leave");` `  gpio_set_level(triggerPinIn,1);` `  vTaskDelay(100 / portTICK_PERIOD_MS);` `  gpio_set_level(triggerPinIn,0);` `  vTaskDelay(100 / portTICK_PERIOD_MS);`	`void leave(){` `  ESP_LOGI(TAG,"Command: Leave");` `  gpio_set_level(triggerPinIn,1);` `  vTaskDelay(100 / portTICK_PERIOD_MS);` `  gpio_set_level(triggerPinIn,0);` `  vTaskDelay(**(ALLOWED_TIME_IN_MIDDLE_STATE-15)** / portTICK_PERIOD_MS);` `  gpio_set_level(triggerPinOut,1);` `  vTaskDelay(100 / portTICK_PERIOD_MS);` `  gpio_set_level(triggerPinOut,0);`

	gpio_set_level(triggerPinOut,1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut,0); vTaskDelay(500 / portTICK_PERIOD_MS); }	vTaskDelay((**ALLOWED_TIME_IN_MIDDLE_STATE+15**) / portTICK_PERIOD_MS); }
3	/** * someone goes to the middle of the doorway, and then turns around * expected count result: no change */ void halfwayEnter(){ ESP_LOGI(TAG,"Command: Half Enter"); gpio_set_level(triggerPinOut,1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn,1); vTaskDelay(50 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn,0); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut,0); vTaskDelay(500 / portTICK_PERIOD_MS); }	void halfwayEnter(){ ESP_LOGI(TAG,"Command: Half Enter"); gpio_set_level(triggerPinOut,1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn,1); vTaskDelay(50 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn,0); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut,0); vTaskDelay((**ALLOWED_TIME_IN_MIDDLE_STATE+15**) / portTICK_PERIOD_MS); }
4	/** * Corner case from Group 4: Almost Enter (big person). * When the student decides to turn around the student has already * unblocked the outer barrier but not the inner one. * expected outcome: no change */ void breaksOuterAndInnerButReturnsG4() { ESP_LOGI(TAG, "Command: breakOuterAndInnerButReturnsG4"); gpio_set_level(triggerPinOut, 1); vTaskDelay(200 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 1); vTaskDelay(200 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut, 0); vTaskDelay(200 / portTICK_PERIOD_MS);	void breaksOuterAndInnerButReturnsG4() { ESP_LOGI(TAG, "Command: breakOuterAndInnerButReturnsG4"); gpio_set_level(triggerPinOut, 1); vTaskDelay(200 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 1); vTaskDelay(200 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut, 0); vTaskDelay(200 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut, 1); vTaskDelay(200 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 0); vTaskDelay(200 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut, 0); vTaskDelay((**ALLOWED_TIME_IN_MIDDLE_STATE+15**) / portTICK_PERIOD_MS); }

	gpio_set_level(triggerPinOut, 1); vTaskDelay(200 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 0); vTaskDelay(200 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut, 0); vTaskDelay(200 / portTICK_PERIOD_MS); }	
5	/** * Corner case from Group9: Almost Enter (slim person). * a person enters the room (breaking the first and then the second light barrier), * turns around (while the second light barrier is still broken), * and leaves the rooms (breaking the first light barrier again quickly after). * expected outcome: no change */ void personTurnedG9() { ESP_LOGI(TAG,"Command: Person entered the room and turned around"); // person entering gpio_set_level(triggerPinOut, 1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut, 0); vTaskDelay(100 / portTICK_PERIOD_MS); // person turning around gpio_set_level(triggerPinIn, 1); vTaskDelay(300 / portTICK_PERIOD_MS); // turning takes time gpio_set_level(triggerPinIn, 0); vTaskDelay(100 / portTICK_PERIOD_MS); // person left the room again gpio_set_level(triggerPinOut, 1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut, 0); vTaskDelay(1000 / portTICK_PERIOD_MS); }	void personTurnedG9() { ESP_LOGI(TAG,"Command: Person entered the room and turned around"); // person entering gpio_set_level(triggerPinOut, 1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut, 0); vTaskDelay((**ALLOWED_TIME_IN_MIDDLE_STATE-15**) / portTICK_PERIOD_MS); // person turning around gpio_set_level(triggerPinIn, 1); vTaskDelay(300 / portTICK_PERIOD_MS); // turning takes time gpio_set_level(triggerPinIn, 0); vTaskDelay((**ALLOWED_TIME_IN_MIDDLE_STATE-15**) / portTICK_PERIOD_MS); // person left the room again gpio_set_level(triggerPinOut, 1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut, 0); vTaskDelay((**ALLOWED_TIME_IN_MIDDLE_STATE+15**) / portTICK_PERIOD_MS); }
6	/**	void unsureEnter() {

	`` * someone almost enters the room, but takes one step back (PinIn goes low before PinOut) * before finally entering. * expected outcome: +1 */ void unsureEnter() { ESP_LOGI(TAG,"Command: Unsure Enter"); gpio_set_level(triggerPinOut, 1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 0); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut, 0); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut, 1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut, 0); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 0); vTaskDelay(100 / portTICK_PERIOD_MS); } ```	``` ESP_LOGI(TAG,"Command: Unsure Enter"); gpio_set_level(triggerPinOut, 1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 0); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut, 0); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut, 1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut, 0); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 0); vTaskDelay((ALLOWED_TIME_IN_MIDDLE_STATE+15) / portTICK_PERIOD_MS); } ```
7	``` /** * Someone is trying to manipulate the count by waving their arm through the barrier towards the inside * Sequence is not possible if a person enters * expected outcome: no change */ void manipulationEnter(){ ESP_LOGI(TAG,"Command: Manipulation Enter "); gpio_set_level(triggerPinOut,1); vTaskDelay(15 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut,0); vTaskDelay(15 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 1); vTaskDelay(15 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 0); ```	``` void manipulationEnter(){ ESP_LOGI(TAG,"Command: Manipulation Enter "); gpio_set_level(triggerPinOut,1); vTaskDelay(15 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut,0); vTaskDelay(15 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 1); vTaskDelay(15 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn, 0); vTaskDelay((ALLOWED_TIME_IN_MIDDLE_STATE+15) / portTICK_PERIOD_MS); } ```

	vTaskDelay(500 / portTICK_PERIOD_MS); }	
8	/** * Corner case from Group11: * Alice peeks into the room, shortly afterwards Bob leaves the room * expected count result: -1 */ void peekIntoandLeaveG11(){ ESP_LOGI(TAG,"Command: Peek into and leave"); gpio_set_level(triggerPinOut,1); vTaskDelay(3000 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut,0); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn,1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn,0); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut,1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut,0); vTaskDelay(500 / portTICK_PERIOD_MS); }	void peekIntoandLeaveG11(){ ESP_LOGI(TAG,"Command: Peek into and leave"); gpio_set_level(triggerPinOut,1); vTaskDelay((**TIME_BEFORE_CONSIDERED_AN_OBSTRUCTION-100**) / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut,0); vTaskDelay((**ALLOWED_TIME_IN_MIDDLE_STATE+15**) / portTICK_PERIOD_MS); //peeking finished //another person exiting gpio_set_level(triggerPinIn,1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinIn,0); vTaskDelay((**ALLOWED_TIME_IN_MIDDLE_STATE-15**) / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut,1); vTaskDelay(100 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut,0); vTaskDelay((**ALLOWED_TIME_IN_MIDDLE_STATE+15**) / portTICK_PERIOD_MS); }
9	/** * successive entering * Alice enters the room while Bob also enters * expected outcome: +2 */ void successiveEnter(){ ESP_LOGI(TAG,"Command: Successive Enter"); // first person entering gpio_set_level(triggerPinOut,1); vTaskDelay(50 / portTICK_PERIOD_MS); gpio_set_level(triggerPinOut,0); vTaskDelay(50 / portTICK_PERIOD_MS); // first person almost inside gpio_set_level(triggerPinIn,1);	Will not work as explained in the comment given above the function

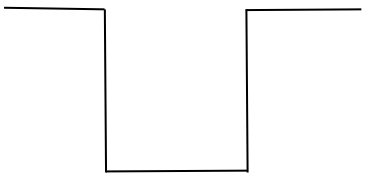
```
    vTaskDelay(50 / portTICK_PERIOD_MS);
    // second person entering
    gpio_set_level(triggerPinOut,1);
    vTaskDelay(50 / portTICK_PERIOD_MS);
    // first person inside
    gpio_set_level(triggerPinIn,0);
    vTaskDelay(50 / portTICK_PERIOD_MS);
    // second person entering
    gpio_set_level(triggerPinOut,0);
    vTaskDelay(50 / portTICK_PERIOD_MS);
    gpio_set_level(triggerPinIn,1);
    vTaskDelay(50 / portTICK_PERIOD_MS);
    gpio_set_level(triggerPinIn,0);
    vTaskDelay(500 / portTICK_PERIOD_MS);
}
```

The above table shows 9 corner cases from the file "*command_corner_cases.c*", but there are adaptation for only 8 of them, as there are 8 cases for which our machine would work properly. In one of the cases from the file, defined by function "*succesiveEnter*" our model would not do as described in the comment above the function definition. The reason for this is because our model was built in a way that it cannot detect consecutive enters (and consecutive exits). Only if one person finished with his interaction with the model (i.e. entered, or exited, or went away from where he came), and then enough time has passed from then, which is the time defined be the "*ALLOWED_TIME_IN_MIDDLE_STATE*" constant, the model could handle the next person interacting with the system.

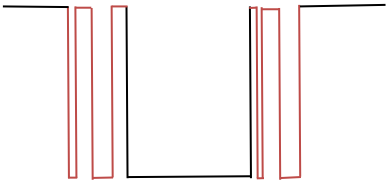
In the case of the provided "*successiveEnter*" function, our model would interpret that instead of the second person entering while the first one was breaking the light barrier on the other side, that the first person simply turned horizontally, breaking both barriers at the same time. Then what would the machine think is that the person changed his mind and completely turned around from entering the room, but then again changed around and went in the room. This would lead to an increase in the counter by one, as the machine would interpret all the rising and falling edges were caused by one person.

We also implemented two measures that to resolve some problems which might occur.

The first one is concerning the problem of bouncing. With help of tracking the time between two signals were registered, we were able to perform debouncing. When we are using mechanical hardware (like the buttons which our group used as a substitution to light barriers), oscillations occur, and cause the signal not to look as someone might expected. On the photo below you see two scenarios, l with the left picture presenting the ideal scenario (the one we thrive to achieve), and the picture on the right presenting a true picture of what is happening in reality.



Scenario 1: How would ideally rising and falling edges look like



Scenario 2: How rising and falling edges look like in reality

For an example, when we push a button (and not let it go), we would expect that only a rising edge appeared, but what instead happens is that some contact bounce appears, and we experience multiple rising and falling edges in a very short space of time. This is similar top what is presented in the “Scenario 2” from the above pair of photos.

We resolved the problem of bouncing using code (although it could be done by using hardware) in a way that when we detect the first rising or falling edge, from that point in the next 8ms (which is a constant that we found on internet that is preferable to use to avoid bouncing) we don’t consider any rising and falling edges.

As for the internet connection, during the last week we faced some serious problems. Our esp32 lost internet connection a few times therefore we lost a few data instances. We dealt with losing internet connection in such a way that our esp32 tries to reconnect until it succeeds which can be seen on the next photo. The count stays the same during this process.