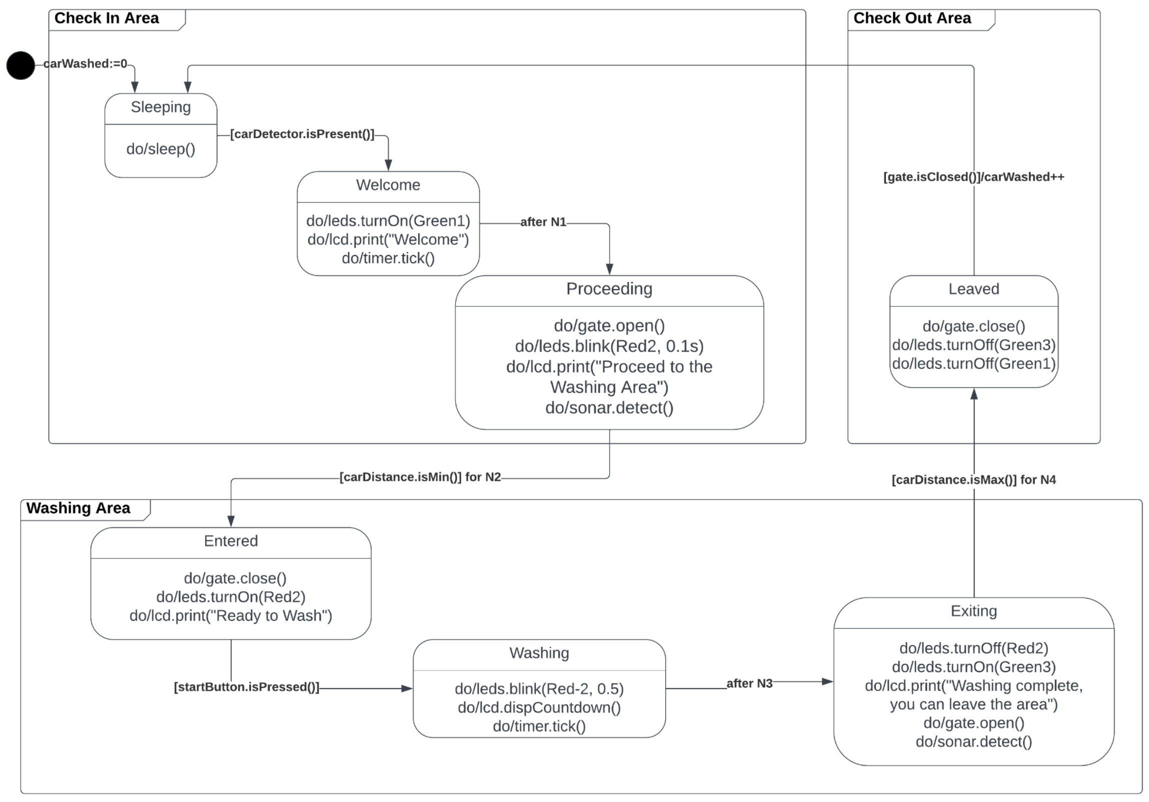
Smart Car Washing

tommaso.severi2studio.unibo.it - tommaso.ceredi@studio.unibo.it

In this document we will be briefly explaining how we decided to tackle a car washing machine made with Arduino.

Design

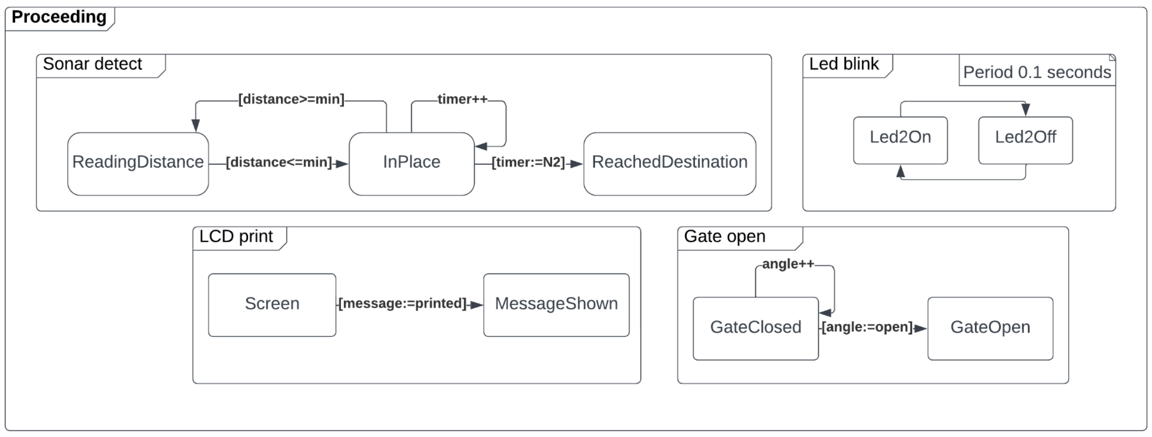
First thing first the architecture of our solution is based on a finite state machine that constantly loops between a predetermined set of states. Every state is executed one by one and each one has to finish for the next to begin: transitions between states are handled by a set of conditions.
In our design we have a total of seven states that are linked to the current position of the car going through the washing machine. Each state is composed of a set of actions that need to be executed before the next state can be triggered.
In the following picture the entire architecture is displayed and each state is encapsulated in the current place the car should be based on the states that are being executed.

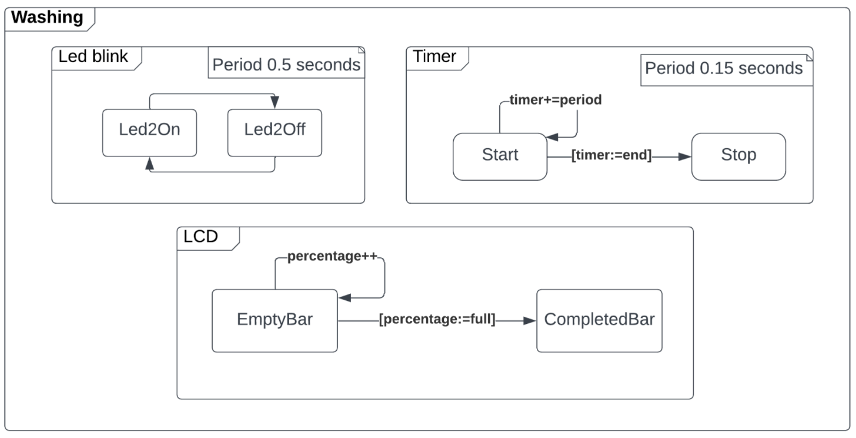


Each state is a task based machine that defines the synchronic nature of the entire system. To further explain this concept we created a few diagrams as an example of what the states are made of. Every task is executed by itself and doesn’t interact or change the state of the other, but the union of all of them is what defines the state that machine is in.

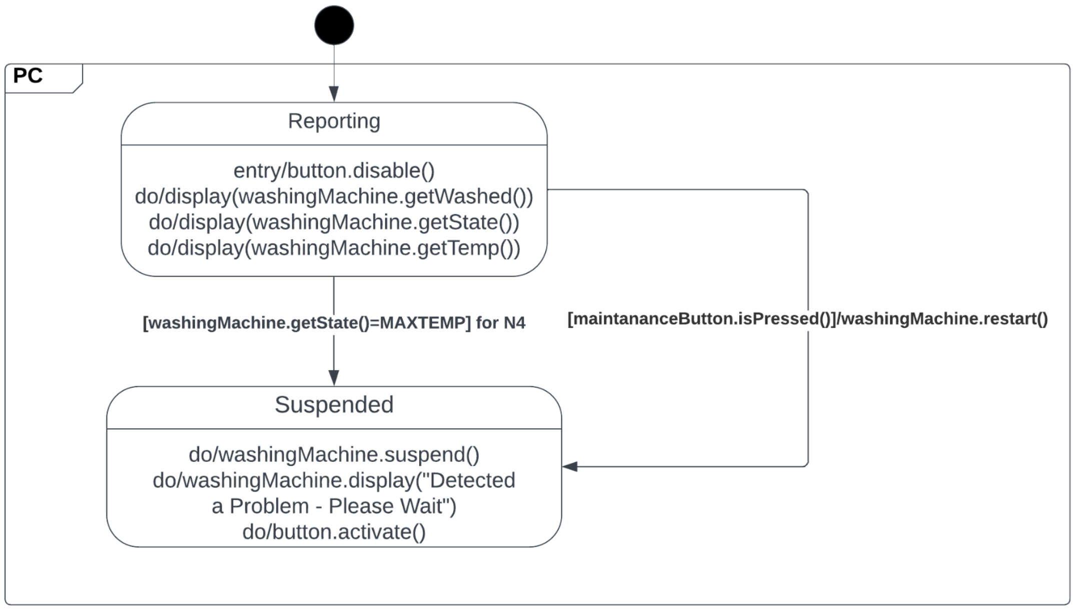
In our example the behavior of a few components are displayed:

- The gate has to increment its angle before it can be perceived as open.
- The Led turns itself on and off with a specific period.
- The lcd either prints a new message or displays each time a new portion of a loading bar depending on the mode it’s on.
- The timer increments its built-in variable of the period that the task is called until it reaches the set time.
- The sonar detects if the car is in the wanted distance and, if it stays there for a specific time, notifies that the car has left/reached the destination. If the car moves outside the range the task resets to the first state.





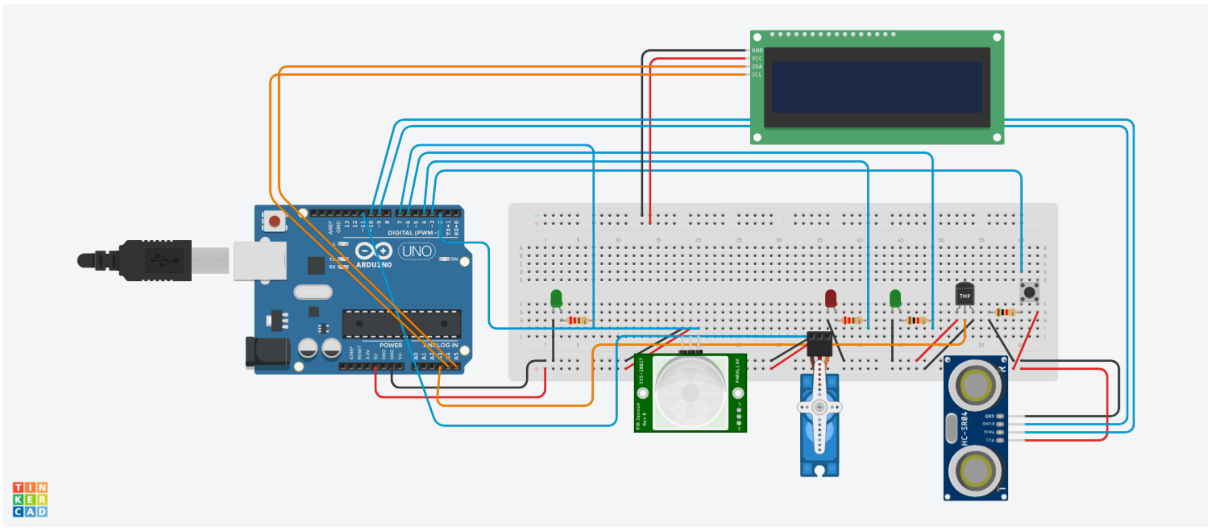
The last diagram is reserved to the part of the project that is intended to be ran on a pc. We also modeled this program to be a finite state machine, it loops between its reporting state that displays the data taken from the arduino and the suspended state in which a temperature error has been reported and needs to be resolved.


PC
Reporting
entry/button.disable()
do/display(washingMachine.getWashed())
do/display(washingMachine.getState())
do/display(washingMachine.getTemp())
[washingMachine.getState()=MAXTEMP] for N4
[maintananceButton.isPressed()]/washingMachine.restart()
Suspended
do/washingMachine.suspend()
do/washingMachine.display("Detected a Problem - Please Wait")
do/button.activate()

Implementation

In this section it is explained how we actually implemented the code to embody the architecture that we created.
The entire program is synchronic and the only element present in the main loop is the scheduler that handles all the tasks that need to be executed during the various states in which the machine finds itself in.
This is obtained by creating a task handler for each state that was defined in the diagram: these classes are the ones that actually contain the tasks of the state and handle everything that concerns them and the state transitions. Each task is then left to directly interface with the hardware.
After initializing all of the task handlers in the setup, each and every task is then taken from them and inserted in the list of the scheduler, but despite that not all of them are executed at the same time; this is the reason why an additional task is fundamental: the state handler.
The state handler is responsible for transitioning between a state and the next. Every task handler has the reference of a method of the state handler and as soon as it's called, from an interrupt or directly, the next handler is put to the execution. This change is not bound directly to the handlers or the states but only puts in execution the next object that it finds in a given list. The actual order of the machine states is actually maintained by the order of the list where the task handlers are put; the order is the

same shown by the state diagram. The state handler being itself a task is fundamental to ensure that the transitions are not triggered during another task.
The scheduler then has in its list the state handler in the first position and checks it every time before going into the other tasks. The fact that not every remaining task is executed at the same time is due to the existence of a sliding window of indices: one marks the start and the other the end. The window is then moved by the state handler of a quantity equal to how many tasks are present in the next task handler list.
All of this to ensure that only the tasks of the current state are executed during its time and are changed for the ones of the next state solely when prompted by the change conditions.
The suspended state that is defined in the pc program’s state diagram is actually handled, in part, by the arduino. When washing a car the temperature is monitored and, when it reaches a temperature too high for a specific time, a warning is sent on the serial line that is then intercepted by the pc program. When the error is then resolved a new message is sent, this time for the pc to arduino, saying that the execution can be resumed. In the meanwhile the arduino waits for this message to be received and stops its execution until prompted to resume; when that happens it restarts the washing process.
Hardware components are maintained by a single class and are initiated only in the setup and then shared by all the tasks.
To summarize the solution: the program has a set of states that are represented by a group of tasks that are then executed, when given their moment, by the scheduler and one special task is used to handle the state transitions.
Next is depicted how we configured the arduino breadboard.


UNO
ARDUINO
DIGITAL (PWM~)
POWER
ANALOG IN
HC-SR04
TMP
PIR Sensor