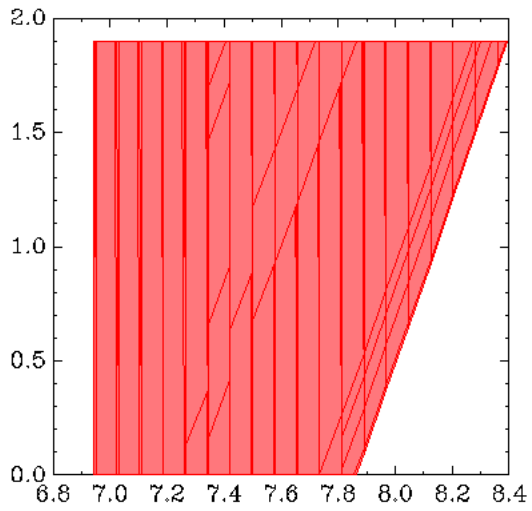
PHAVER Report by Ali Dhanani 000470296

While working with the phaver file, I had used the binary file "phaverlite-0.2.2_static" in the source folder. The process of running the file is described having in mind the binary file being in the source folder.

Task1: The task was about to model the "bike automaton sensing the FR signal". By analyzing the provided figure in the project file, I have implemented the state **Active** and **Inactive**. Furthermore, as specified that we have to enhance the provided model by completing the **Inactive** state as in the figure it shows that it is empty. By reading where it says, *"inactive control state where the motor assistance is turned off, leading to the existence of only user force U"*. I used the idea of transition from **Active** to **Inactive** which the sync of **stopM**. By following the document's figure, I made a transition from **Active** to **Inactive** with the sync of **stopM**. This has led me to an **Inactive** state. As the document stated that in **Inactive** state there is only user force, so we will put this formula to be there while it is still in **Inactive** $d^{\cdot} = v$, $v^{\cdot} = U - kv/m$. Notice here we have U instead of 2U, as it stated in the document that 2U are there because "a force U, and an additional force of the same amount is provided by the motor assistance, making the total force equal to 2U", because there is no motor assistance force so we just used U instead of 2U. Also, we have to get back to **Active** from **Inactive**, the sync of **startM**, it will move back to the **Active** state. Also, you will find the invariant in the active and the inactive state to be $0<=v<10$, where I have defined the lowest value of the velocity to be 0 and the max value to be less then 10. The sync **stopM** and **startM** are controlled by the controller that we will see in the next task.
The file is provided in the folder. It will be executed as follows:
./phaverlite-0.2.2_static DhananiTask1.pha

Task2: The task firstly was to model the figure "controller for bike for switching off the motor". In the file DhananiTask2.pha, There are 2 models, 1 is the sensor that we had made in Task1 and the other model is called controller that we made in Task2 while analyzing the figure 6. Also, we had made some modification in the model sensor, where we have added an error state as instructed in the Task. The idea is that while being in the **Active** if at any point there is an increase of the velocity from (125)/18, it will move to the error state and will remain there as it has violated the speed rule.
It is executed as follows:
./phaverlite-0.2.2_static DhananiTask2.pha

When here the value of delta is set to $\frac{1.9\times18}{125}$, we can see the graph that shows the velocity being 6.9 when it reaches the d at 1.9. We can see the linear increase after the velocity 7.8, having a linear increase till the velocity 8.4. With this delta value we can say that the motor goes above 6.9 m/s velocity value. This is the graph of d by v. The idea over here was to find the minimum value of delta for which there is no violation of the law of not going above 125/18 m/s. So, by applying binary search we found the value to be 0.3726, if we exceed this delta the graph will be empty.
We obtain the following graph by having the delta value as $\frac{1.9\times18}{125}$:

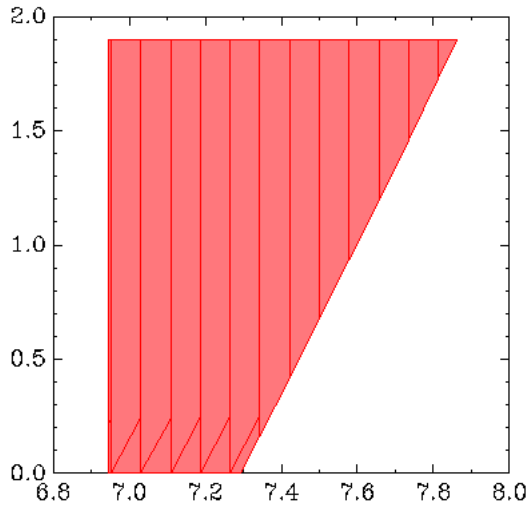


Task3: The task has used the same model as in task2, but with a little modification that are as follows, we have added a delay before going to the off state or the on state. The delay is of 0.5 which is a fix value set by the variable name delta2. The idea is that when we are in state **Stop** instead of checking with the delta1 value, now we will be checking with the delta2 value the clock x. So, the clock value x in increase and we check when the clock value is equal to delta2 value then we perform a sync of **stopM** to go from **stop** to **off**. While we are in the **Stop** state, we modify the clock variable by increasing it by 1 and 0.5. Also, as the document says there can be **FR** signal at any time, so we placed a guard as anytime there can be a signal of **FR** that will make the clock value back to 0. We perform the same operation for **startM** where it is going from **start** to **on**. We place a condition on **start** state, as when we are there the x clock value increase by 1 and 0.5. Plus, the only time it can move from **Start** to **On** when there clock value is equal to delta2, once it is equal to delta2, the sync signal **startM** is made. Also, here the FR signal can be produced anytime, so a guard is placed that any time a FR signal is produced it move to the same state again making the clock variable x as 0.

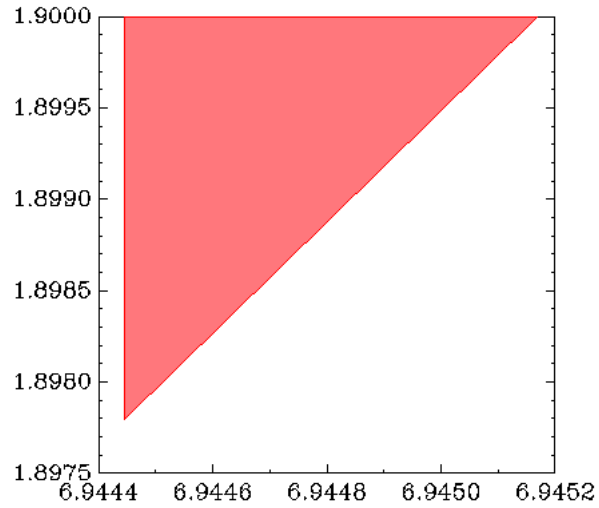
It is executed as follows:

./phaverlite-0.2.2_static DhananiTask3.pha

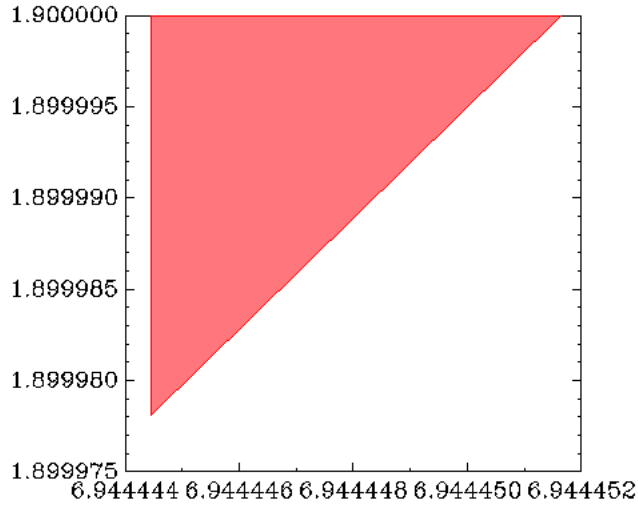
We can observe the following graph, where we can see the graph width of 6.9 to 7.3 unlike in task 2 where we had the width of 6.9 to 7.9. Here also we can see that when have delta value $\frac{1.9\times18}{125}$, it exceeds the velocity of 125/18 m/s. So we need to find the delta1 value that does not allow to violate the speed limit, the delta1 value here will be different then that of the **Task2** as here we have the latency period.



d) Delta value for task 2 is 0.3726, as after this we see the state being empty.



Delta value for task 3 is 0.322743, as after this we see the state being empty



The delta value for task 2 is bigger which is 0.3726, I believe because there doesn't exist a latency or delay in the model like it does in task 3. Taking the idea of stopping, when we apply the stop the motor will not directly go to 0, there will exist still a little speed increase

that will cause a little increase before going to 0, which is what has happened in the task2 with the delta value that was found. In task3, we think about the delay period from the applying the stop and making it 0. By considering the latency period there, we had fixed the delta2 value to be 0.5 in state start and stop before going to the on and off state to make it work in reality considering the factor of an actual delay between the action and implementation. Before in task2 where we didn't take this consideration into account, the delay was there and wasn't being handled whereas in task3, we handled this delay by which, it can be seen that the delta value that we found in task3 is less than that of task2 because of the account of the latency.

e) The motor in task2 and task3 produces the FR signal that are required in the Luster part of the code for dealing with motor failure and motor anomaly. Also, the idea is that in the part of the controller we are dealing with having the delta value in taking account the latency, that the motor assistance is turned off when the speed in above 7 m/s, which is required in the Luster part 3.3 of Motor assistance where while making the node Assist, we require the condition for the speed to be less then 7 or else the assistance is 0.