Automated guided vehicle (AGV) embedded system design and control implementation*

Ahmed Alrefaie
Mechatronics Department
German university in Cairo
46-5804
ahmednabawy777@gmail.com

Mohamed Ibrahim
Mechatronics Department
German university in Cairo
46-20529
mohamedahmedsalahedin@gmail.com

Sherif Khalid
Mechatronics Department
German university in Cairo
46-10638
sherif.khalid@student.guc.edu.eg

Adham Ahmed El-Serougy
Mechatronics Department
German university in Cairo
46-0537
adham.elserougy@student.guc.edu.eg

Ayman Maged Bassyouni
Mechatronics Department
German university in Cairo
46-15946
Ayman.alsayed@student.guc.edu.eg

I. Introduction

The Automated Guided Vehicle (AGV) is capable of transporting designated objects/products in a closed area such as warehouses in factories or industrial applications to transfer material around a manufacturing unit. Usually, materials and products are transported and moved using workers which leads to a waste of time and increases labor costs. Also, it's unsafe for the workers' health due to the large weights they carry and the repetitive tasks they achieve. Also, workers may by accident expose materials and products to break and scratch. Therefore, AGVs were used to solve the problems related to the traditional industry in terms of carrying products from the shelves of the warehouses to the designated trucks which will carry them to the desired shop or the consumer himself.

Fig 1: Automated Guided Vehicle

AGV Navigation can be mainly split into two different methods:

- Physical Navigation:
  - Line Following Navigation
  - Tag the Following Navigation
  - Wire Guidance Method
- Virtual Navigation:
  - Laser Triangulation
  - Vision Guidance (This is the method that we will be using)
  - Natural Feature Navigation

II. Implemented System

The system implemented for The AGV will be an Isolated System that will disregard all frictional effects, such as Aerodynamics effects, Internal Motor Friction, Wheel Friction, etc. on the AGV Dynamics. As stated earlier, The AGV loads and unloads certain packages into designated stations therefore, the loading and unloading process of The AGV in this project will be done using a robotic arm which will not be part of this project. Along with moving the packages the AGV will concurrently monitor its battery health and proceed to charge when requested by the FSM. The AGV also monitors if a certain error occurs within it and notifies the FSM that an emergency occurred

Fig 2: Robotic Arm loading The AGV.

Using Simulink, The AGV's system will use vision guidance. The inputs will be the movement calculated from the simulated FMS (Flexible Manufacturing System) and inserted into The AGV as follows (System input signals):

- Forward.
- Backward.
- Rotate clockwise.

- Rotate anti-clockwise.
- Wait for the package.
- Brake
- Dock.
- Id

The system states will therefore be defined as:

- On
  - ➢ Battery Management(concurrent)
    - ❖ Charging
    - ❖ Discharging
  - ➢ Active Task(concurrent)
    - ❖ Motion
      - Translation(concurrent)
        - Forwards
        - Backwards
        - No translation
      - Rotational(concurrent)
        - Clockwise
        - Anticlockwise
        - No rotation
    - ❖ Waiting
    - ❖ Going to Station
      - Docking
      - Docked
- Off
- Emergency

Whereas, The Translational and Rotational motions will be concurrent with each other. The AGV's system will output:

- Position.
- Fault LED.
- Battery percentage

The continuous dynamics of the system will be modelled as the motion resulting from 2 electrical motors and will be controlled using raspberry pi. The raspberry pi we will be using is raspberry pi 4b 4 GB RAM.

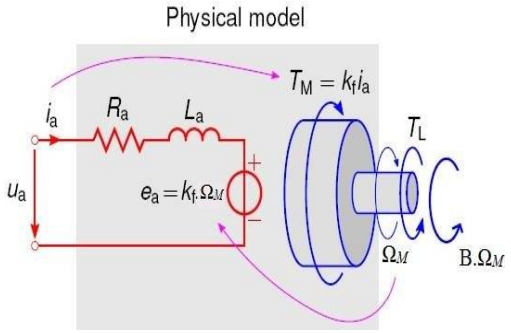
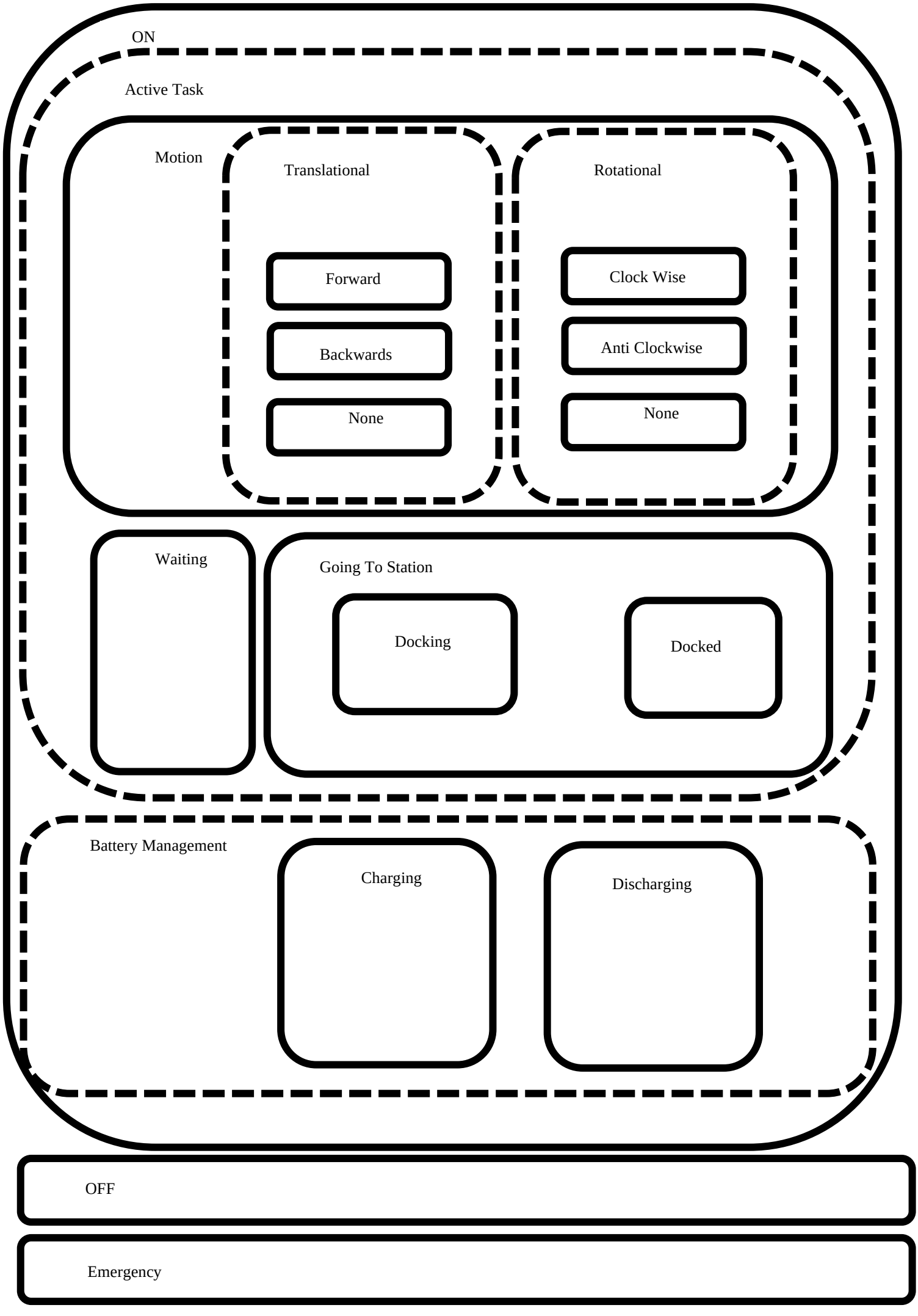
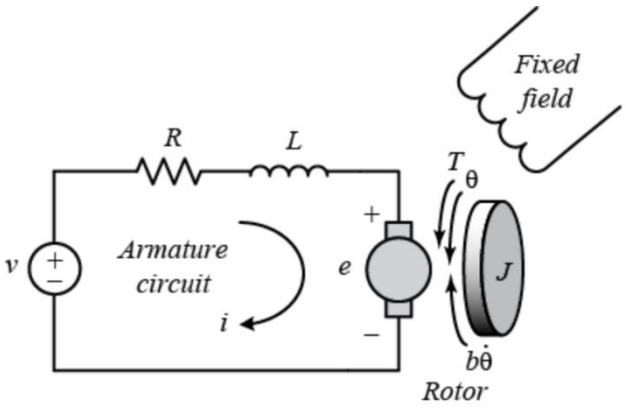


Fig 3: Electrical motors model
(Photo from Enhancement of the Dynamic Performance of a DC Motor using Fuzzy Logic Algorithm)

Fig 4: Raspberry pi.

Fig.4:state chart





As for our EOM this is the model for the used DC motor. Assuming fixed magnetic field the Generated torque will be directly proportional to the current by a constant Kt

$$T = K_t\, i(t) \qquad (1)$$

The back emf is proportional to the angular velocity of the shaft by a constant factor k_e

$$e = K_e \dot{\theta} \qquad (2)$$

In SI units, the motor torque and back emf constants are equal, that is $K_t = K_e$ therefore, we will use K to represent both the motor torque constant and the back emf constant.

Next, we will apply Newton's law and Kirchoff's law to the motor system to generate the following equations:

$$J\ddot{\theta} + b\dot{\theta} = \mathrm{Ki} \qquad (3)$$

$$L\frac{di}{dt} + Ri = V - K\dot{\theta} \qquad (4)$$

State-Space representation:

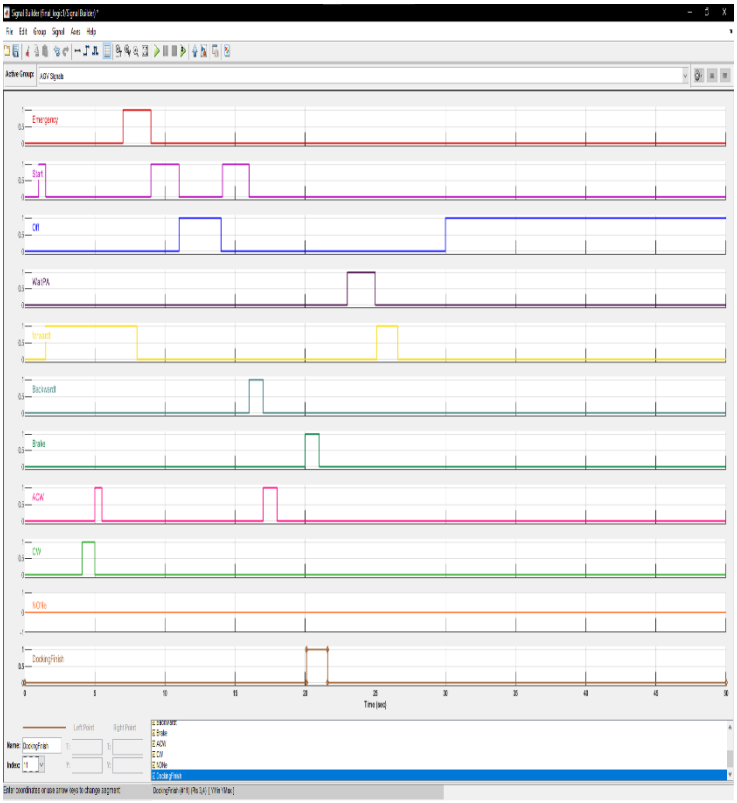
$$\frac{d}{dt}\begin{bmatrix} \dot{\theta} \\ i \end{bmatrix} = \begin{bmatrix} -\frac{b}{J} & \frac{K}{J} \\ -\frac{K}{L} & -\frac{R}{L} \end{bmatrix}\begin{bmatrix} \dot{\theta} \\ i \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} V$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix}\begin{bmatrix} \dot{\theta} \\ i \end{bmatrix}$$

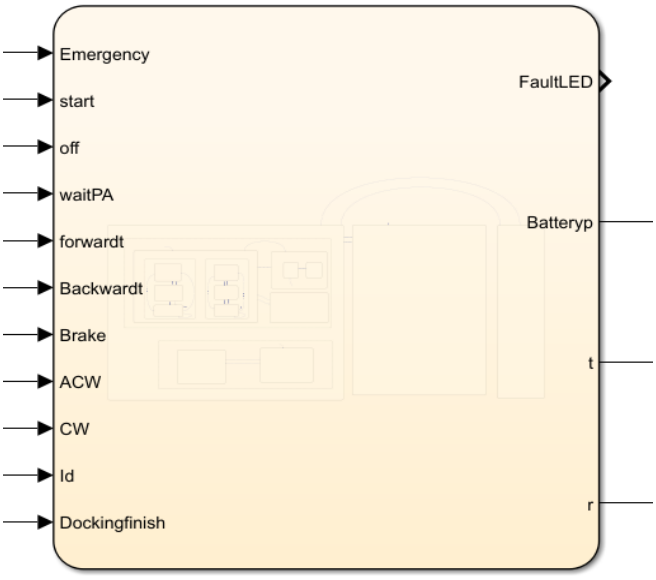
The output of the state chart block here contains the output of the discrete logic (r and t) which are indications of movements of the AGV. E.g: t=1 translate forward and t=-1 translate backward. This will be input to the continuous dynamics block to transform this to Wa and Wb (angular velocities of motor A and B) and Va and Vb (velocities of motor A and B).

Results:

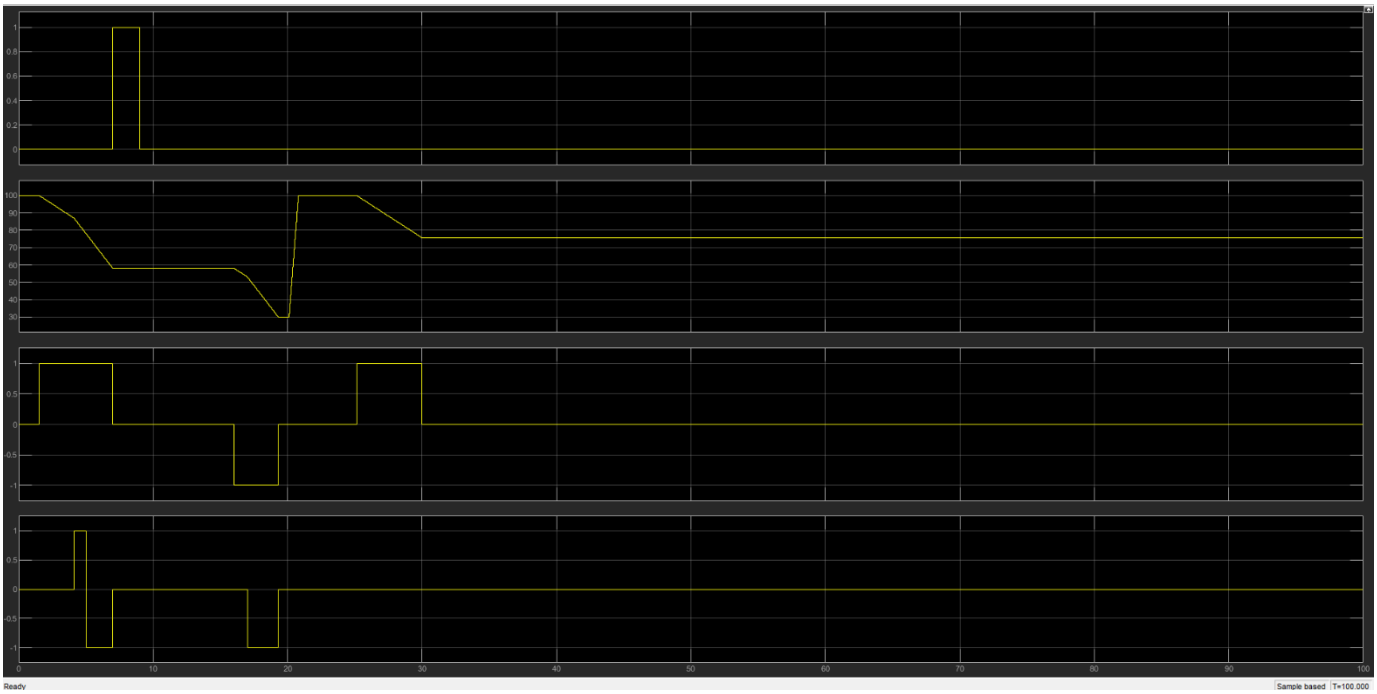
The inputs were generated using a signal builder using different inputs and cases.



The 1st one is Emergency and the inputs have the same order as the inputs of the state chart.

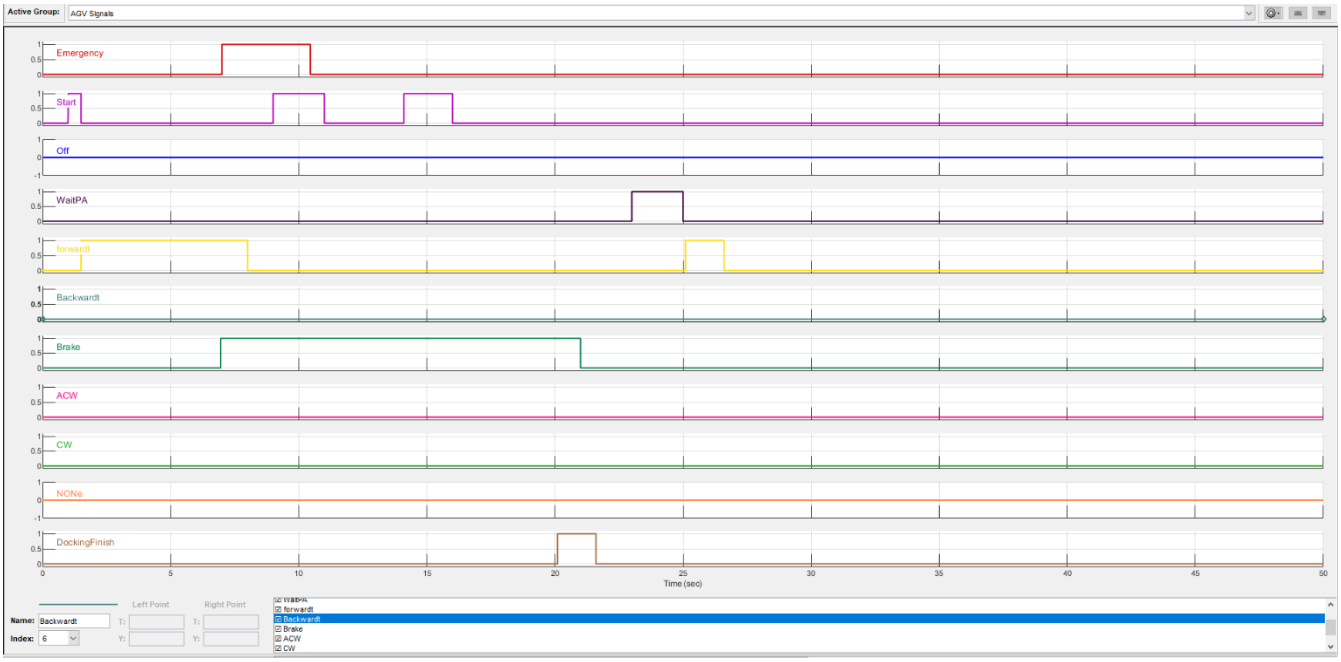


The results for the simulated inputs with different cases:

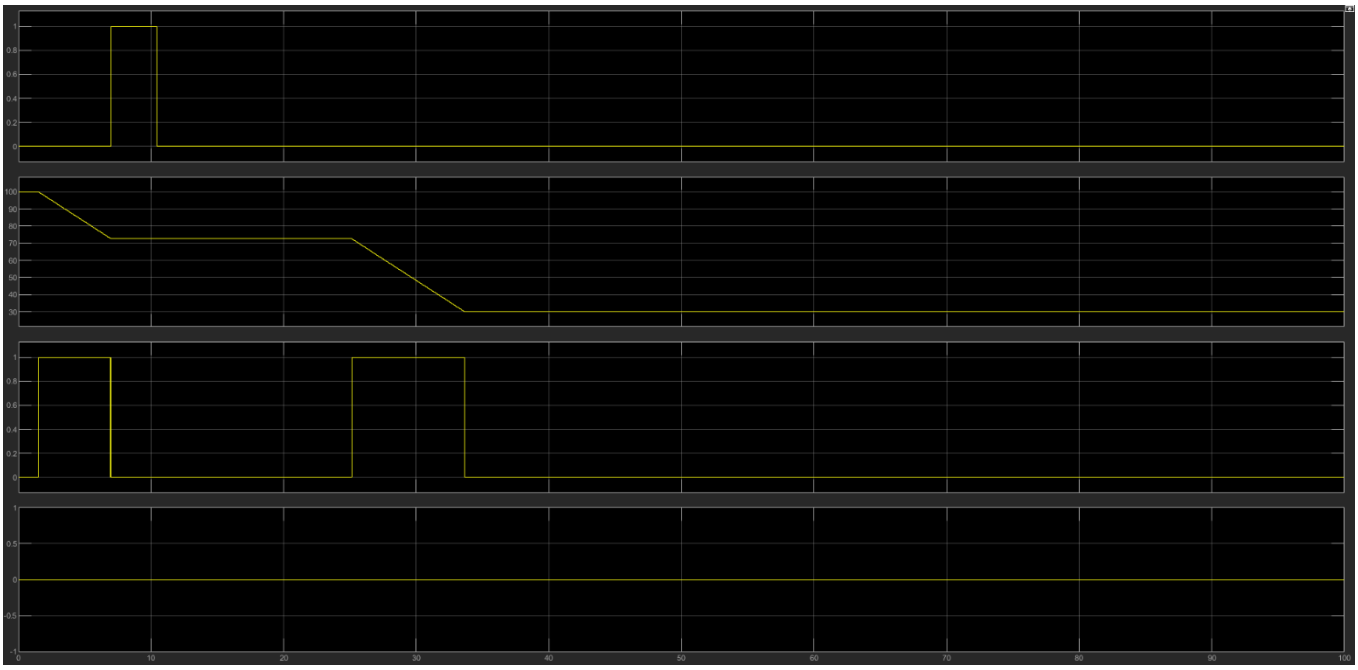


As we can see here the output LED turned off when the Emergency was = 0 and the start was 1 as desired. Also, The Batteryp behaved as desired dor charging and discharging fases.

Another case:



Outputs:

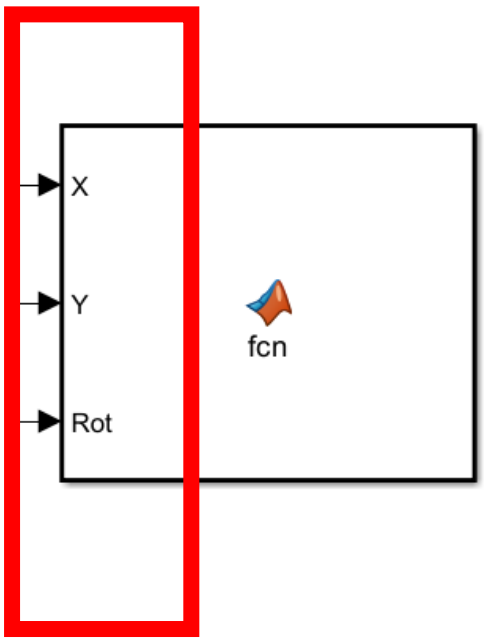


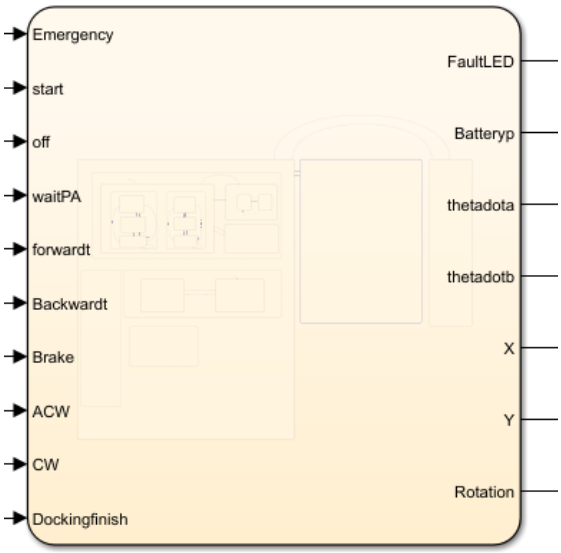
The charging behaved differently because the inputs was translation only while the 1st case was translation and rotation this is why in the 1st case it was discharged faster while in this case it discharged with a slower rate. When the movement stopped no discharging happened.

III. Design of Hybrid state chart

The state chart implemented earlier had r and t as outputs. Thus, implementing continuous dynamics and a controller inside the state chart would mean changing in the inputs and outputs of the state chart in a way to match our designated system design. The state chart now outputs X and Y and Rot and are given as inputs to a MATLAB function to plot and simulate the movement of the AGV.

The inputs of the state chart now becomes:





It is slightly different because the Id now is considered a local variable not as an input anymore due to the designing of the continuous dynamics inside the state chart.

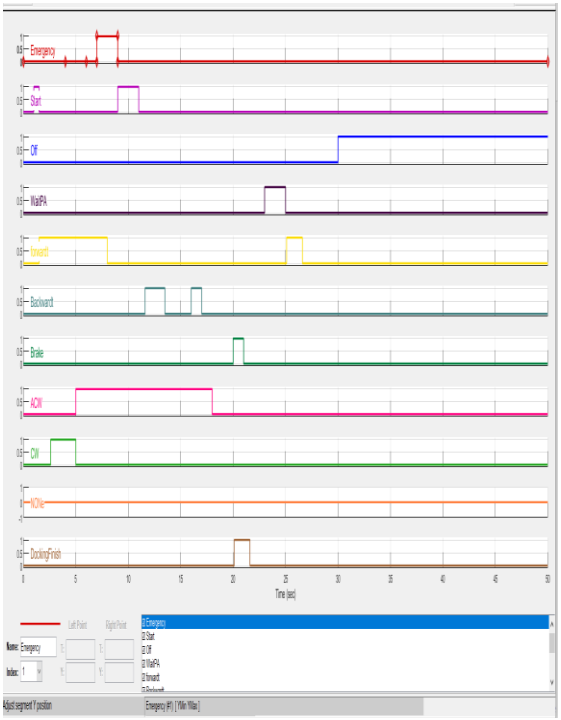
The continuous dynamics as explained earlier had several constants. The following list will be the constant and the corresponding value implemented in our system.

- J=0.01
- b=0.1
- Ke =0.01
- Kt=0.01
- R=1
- L=0.5
- Ts = 0.01 (sampling time)

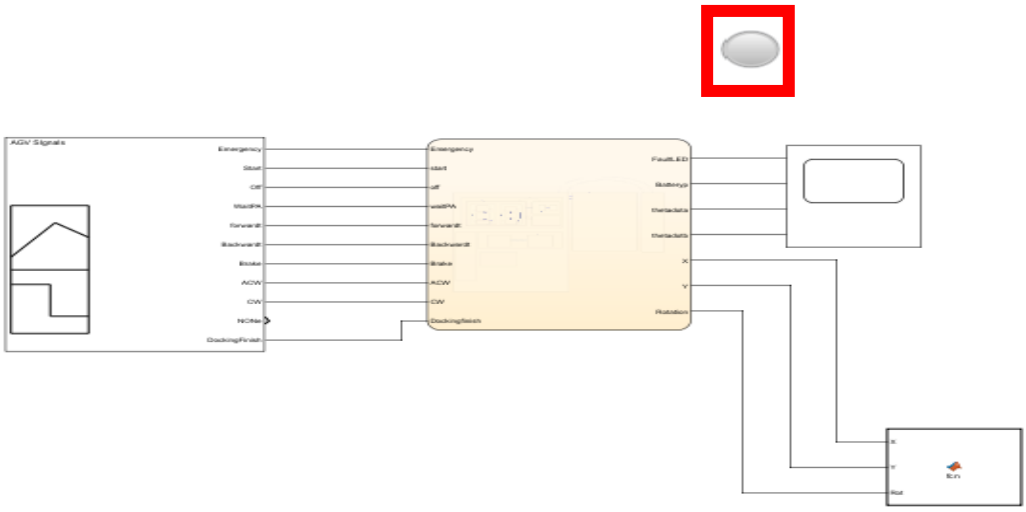
The output of the continuous dynamics is the angular velocity of motor A and B. The angular velocity is then given as feedback to the PID controller. Use of PID controller is valid in this case due to uncoupling of the dynamics of the system. The gains used for the controller are:

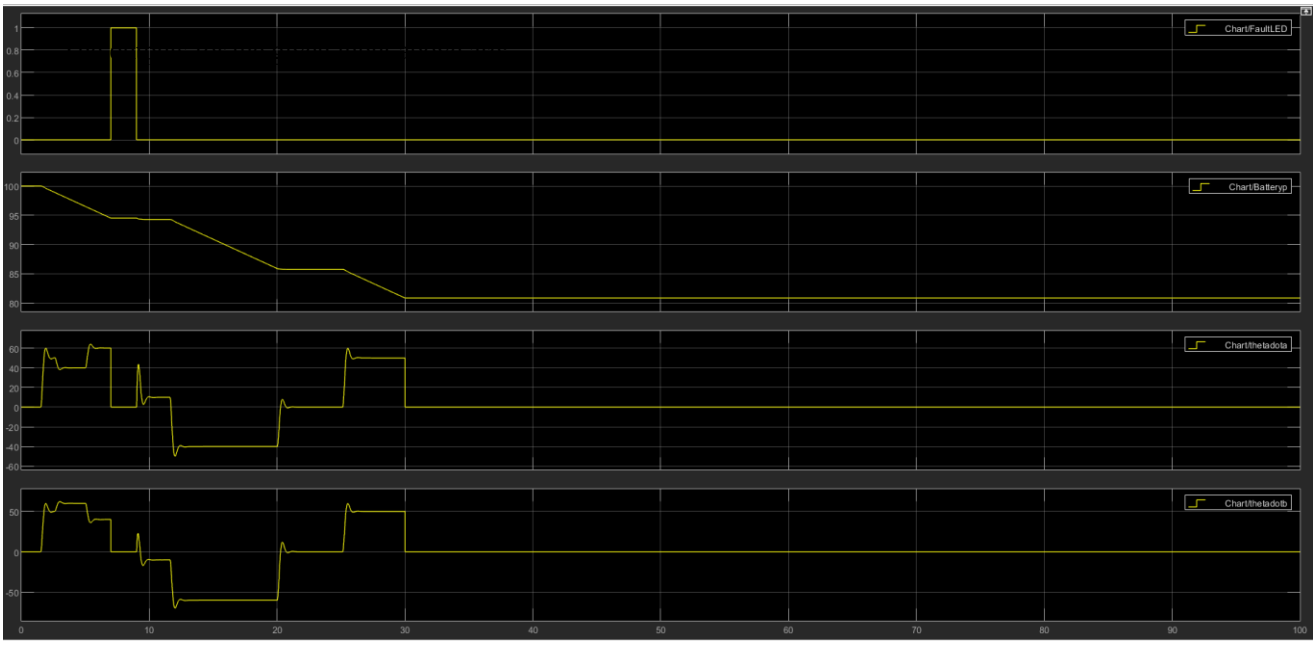
- Kp=40
- Kd=0.01
- Ki=100

The inputs are then input as a signal builder to observe the outputs and effect the controller.



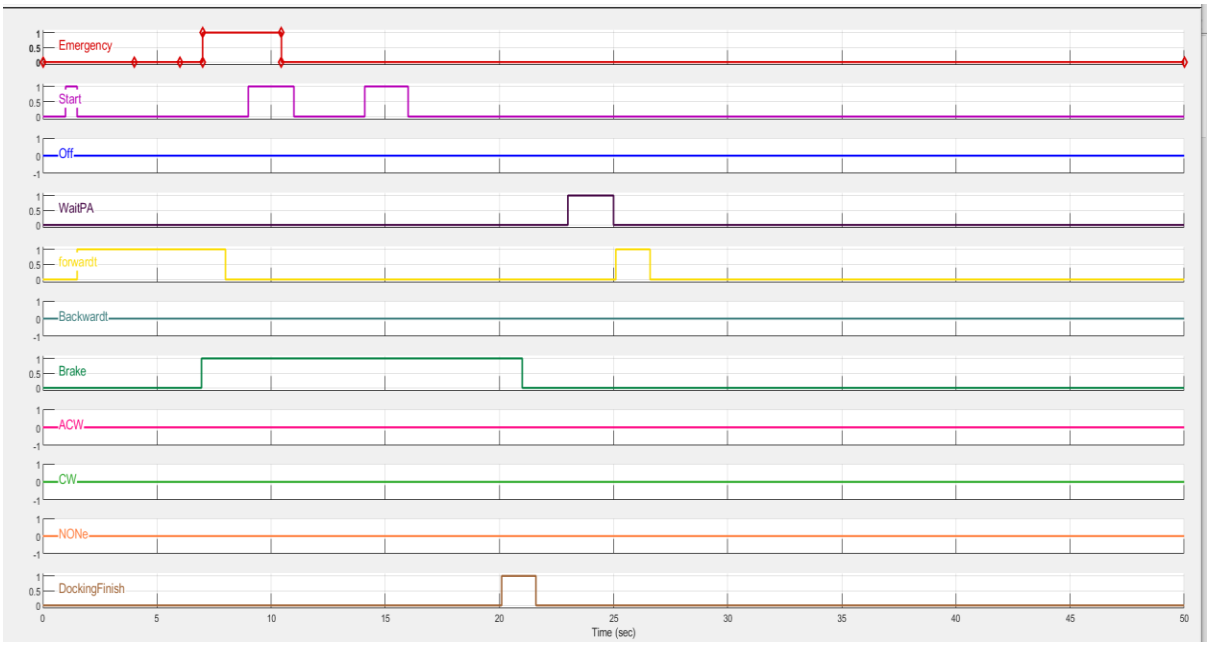
A fault LED in used in Simulink to represent the emergency case.



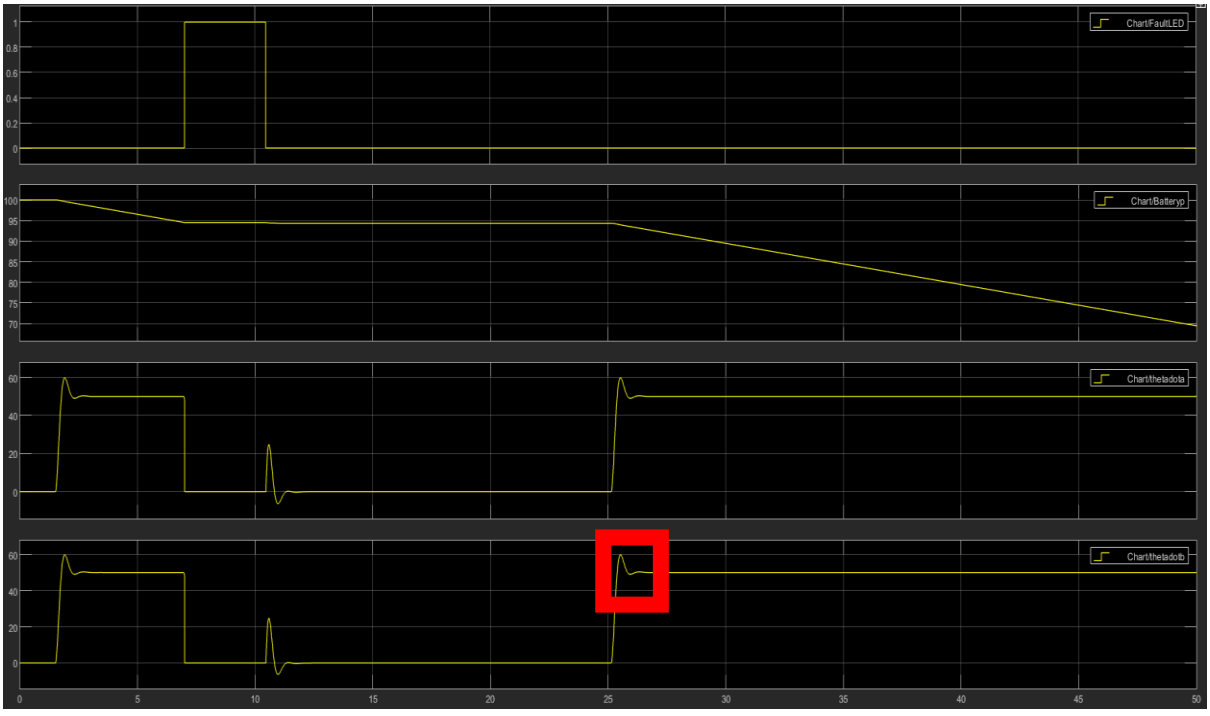


The outputs here gives a really good sense of the movement and dynamics of the AGV with respect to the inputs given. The fault LED signal is shown and also the battery percentage which decreases along movement. (no need of recharging because battery percentage did not reach 30). Also, the angular velocity of A and B are changing due to change of inputs. Translation then CLCK_WISE then Anticlockwise etc.

The 2nd set of inputs were:



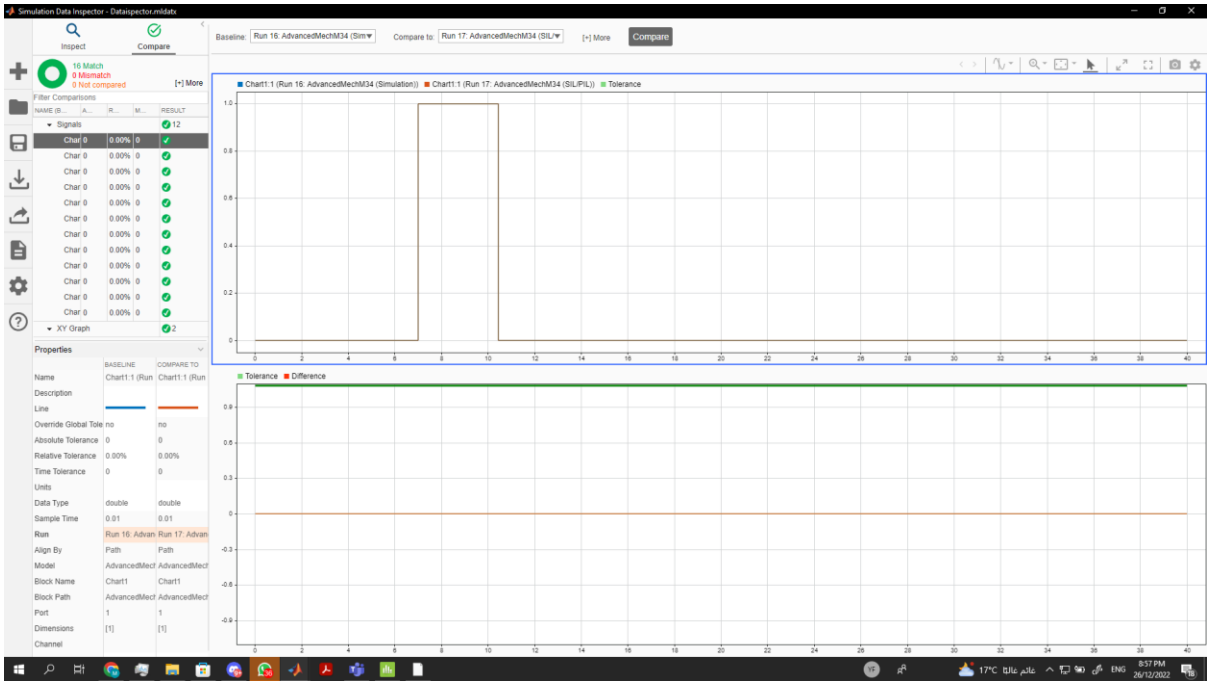
The outputs were:



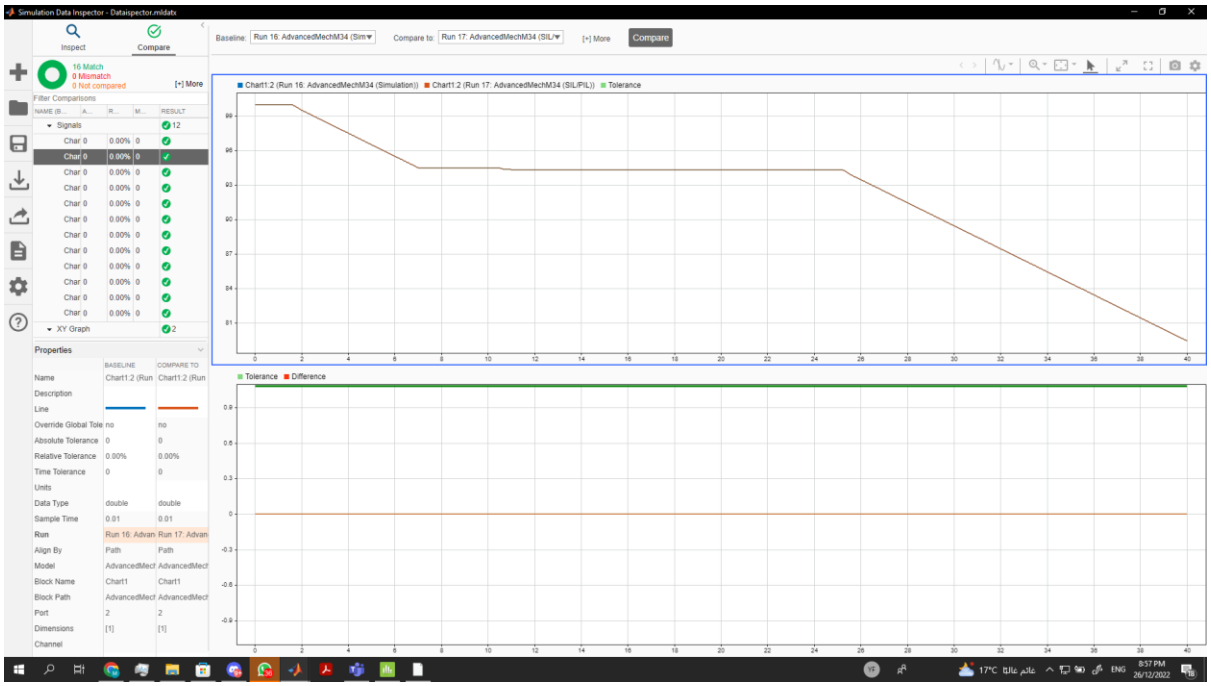
We can see here that the overshoot resultant from instant change of direction is relatively low due to the control gains applied in the PID controller.

IV. Hardware in the Loop:

A. *Simulation*



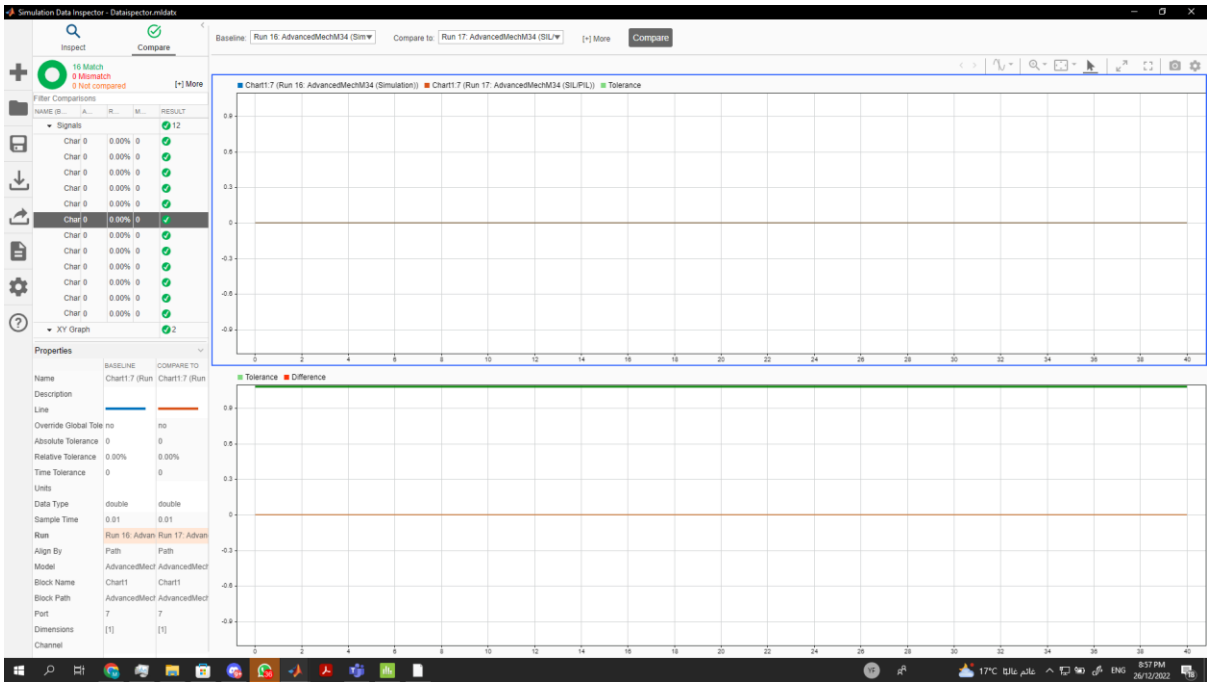
Fault LED Simulation



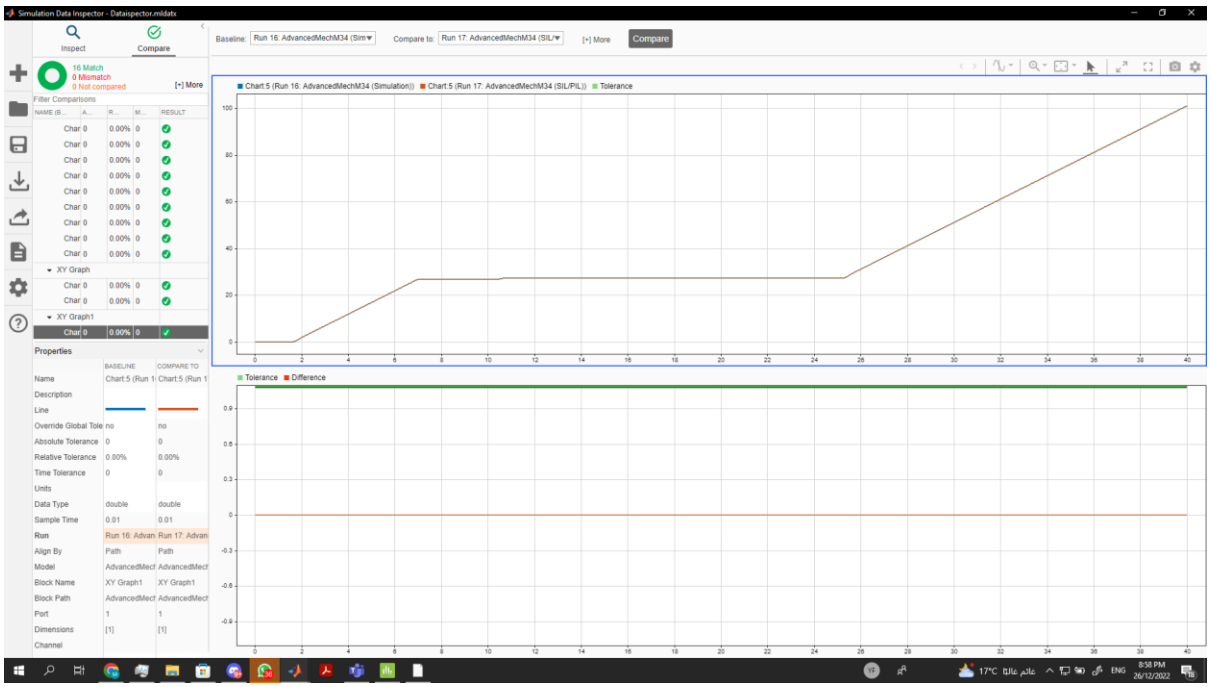
Batteryp Simulation

Thetadota Simulation

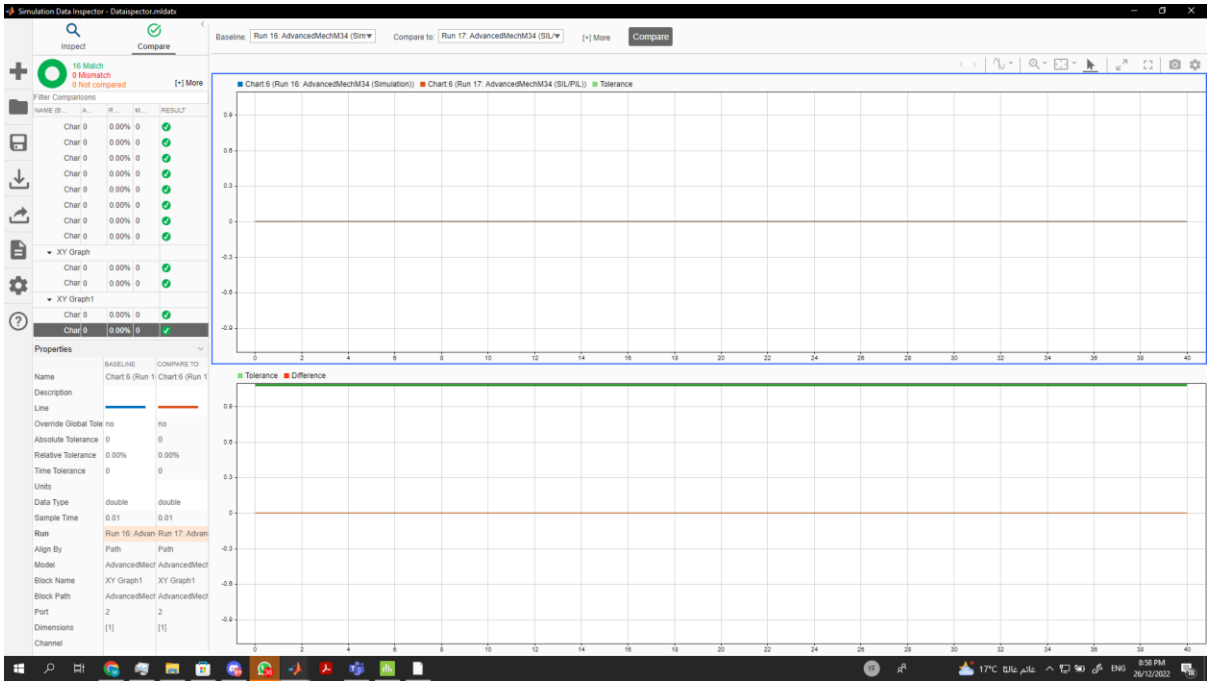
Thetadotb Simulation



Rotation Simulation

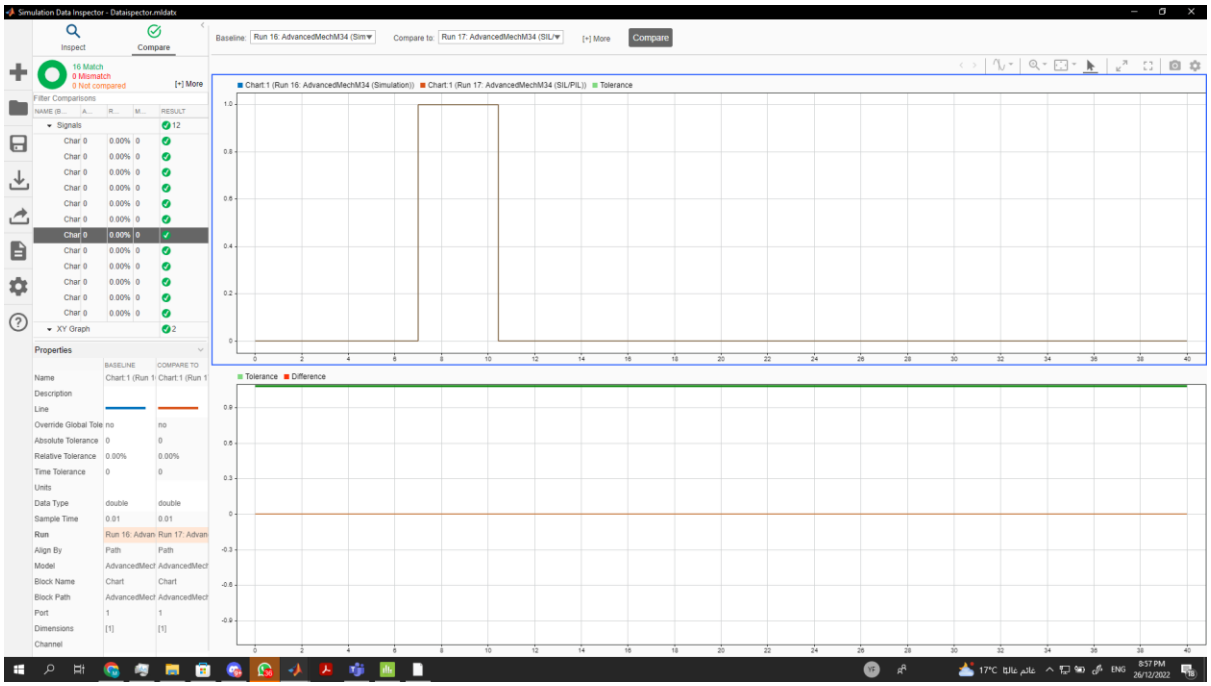


X Simulation

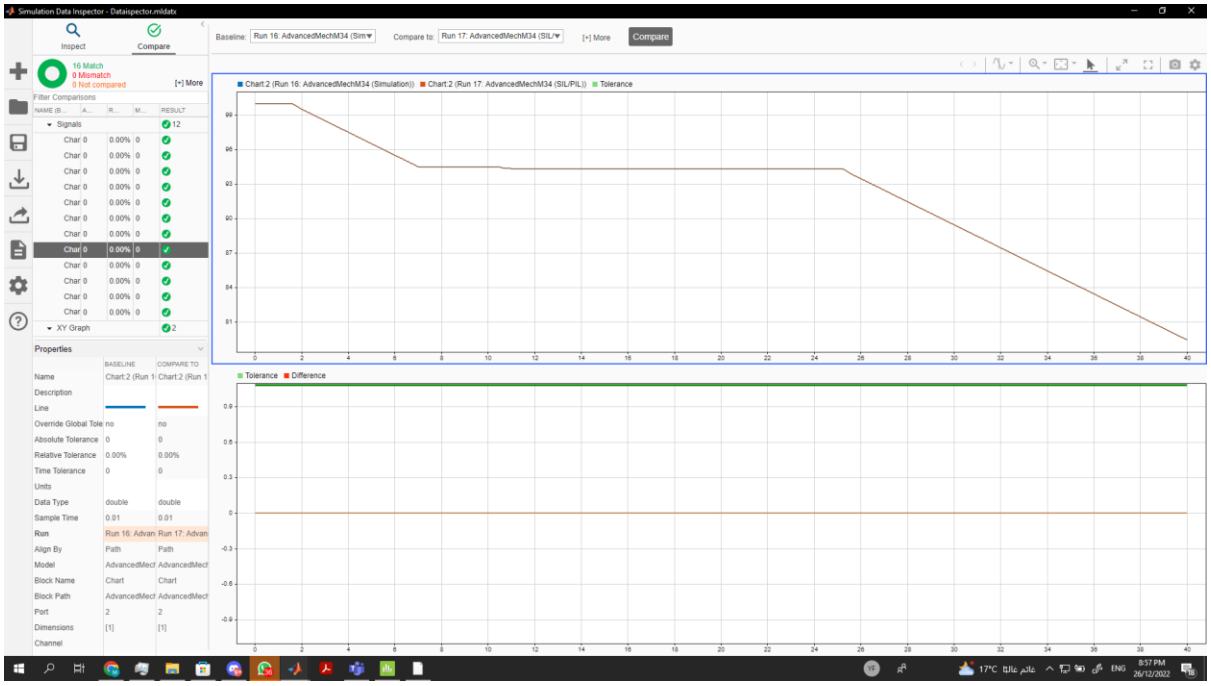


Y Simulation

B. PIL



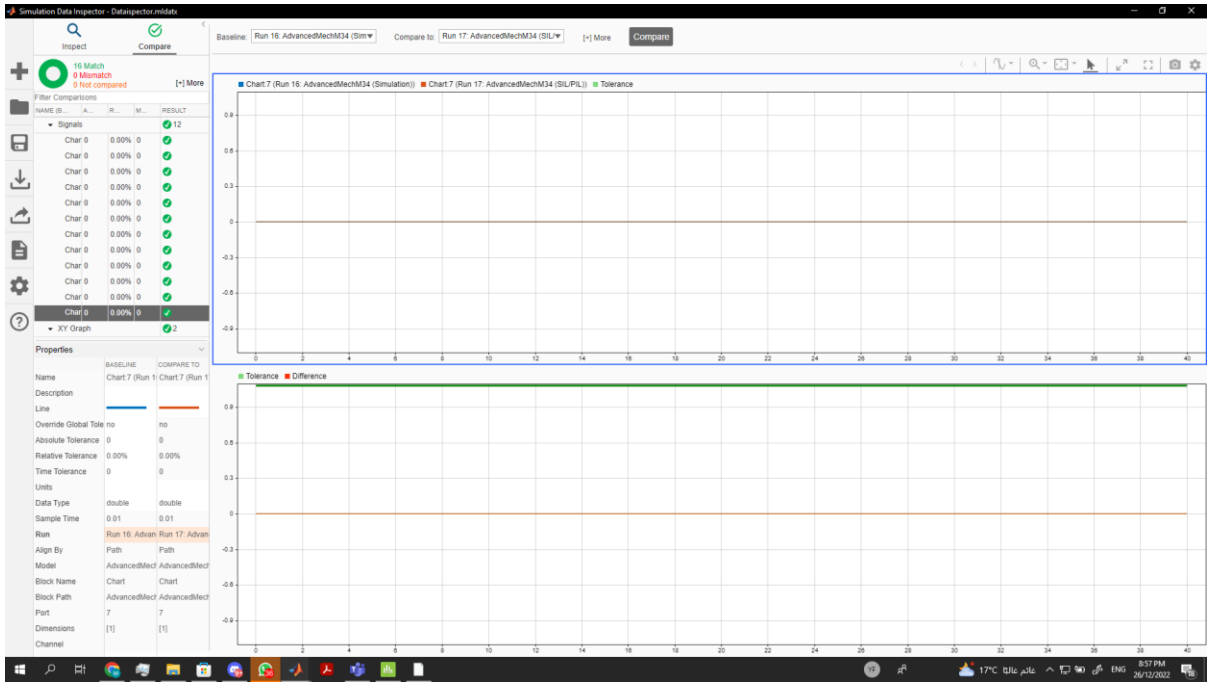
FaultLED PIL



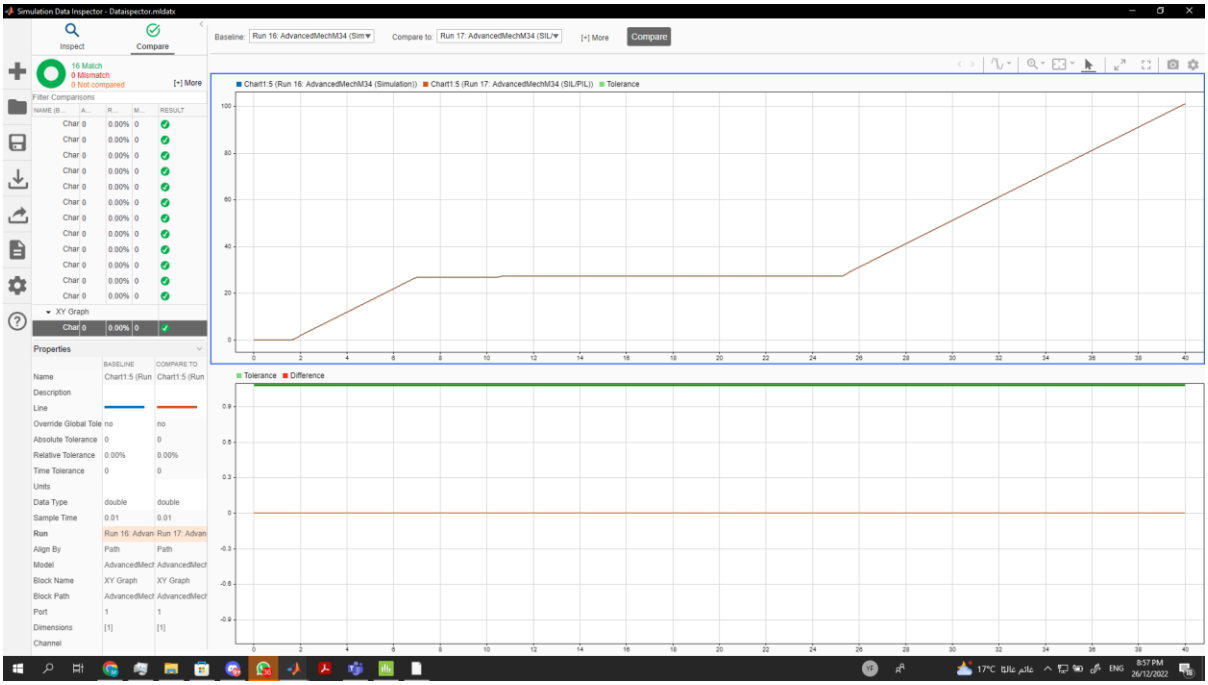
Batteryp PIL

Thetadota PIL

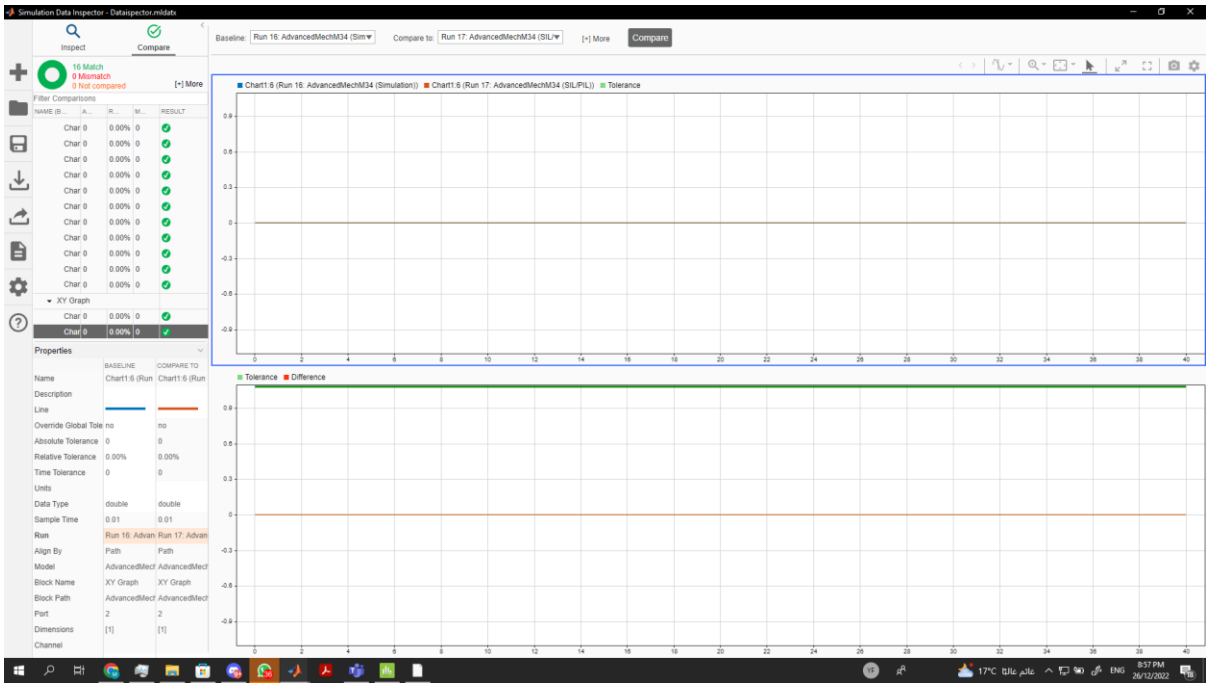
Theatdotb PIL



Rotation PIL

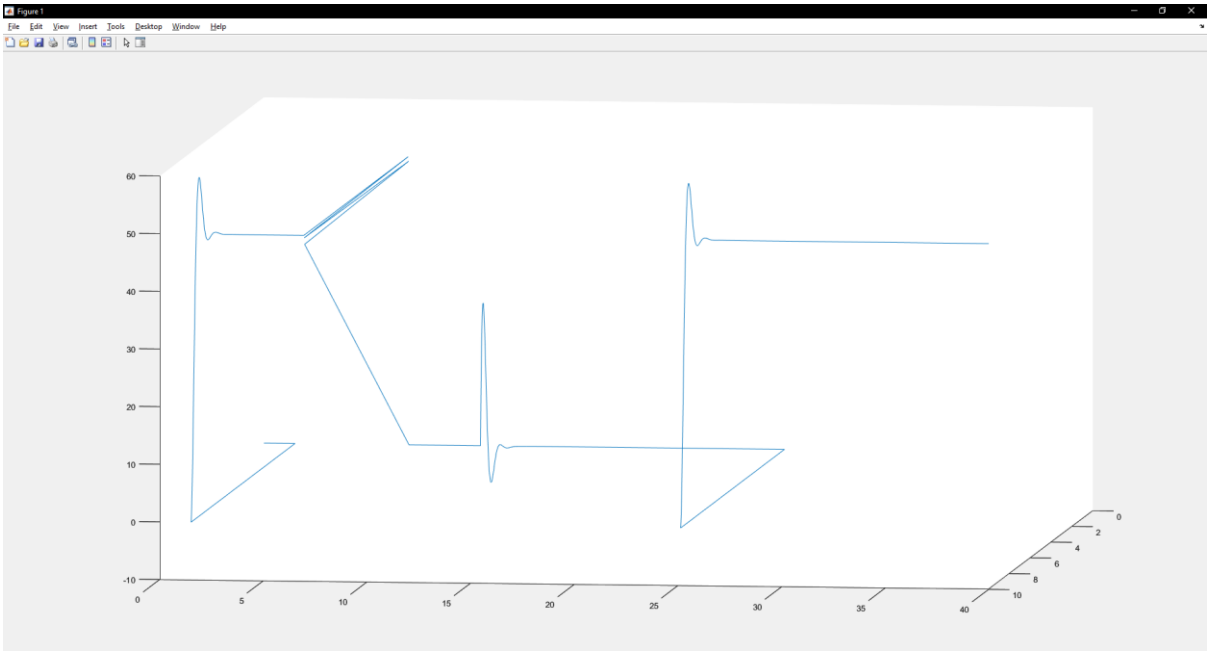

Simulation Data Inspector - Dataispector.mldatx
Inspect
Compare
Baseline: Run 16: AdvancedMechM34 (Sim
Compare to: Run 17: AdvancedMechM34 (SIL/
[+] More
Compare
16 Match
0 Mismatch
0 Not compared
Filter Comparisons
Properties
Chart1:5 (Run 16: AdvancedMechM34 (Simulation))
Chart1:5 (Run 17: AdvancedMechM34 (SIL/PIL))
Tolerance
Difference

X PIL

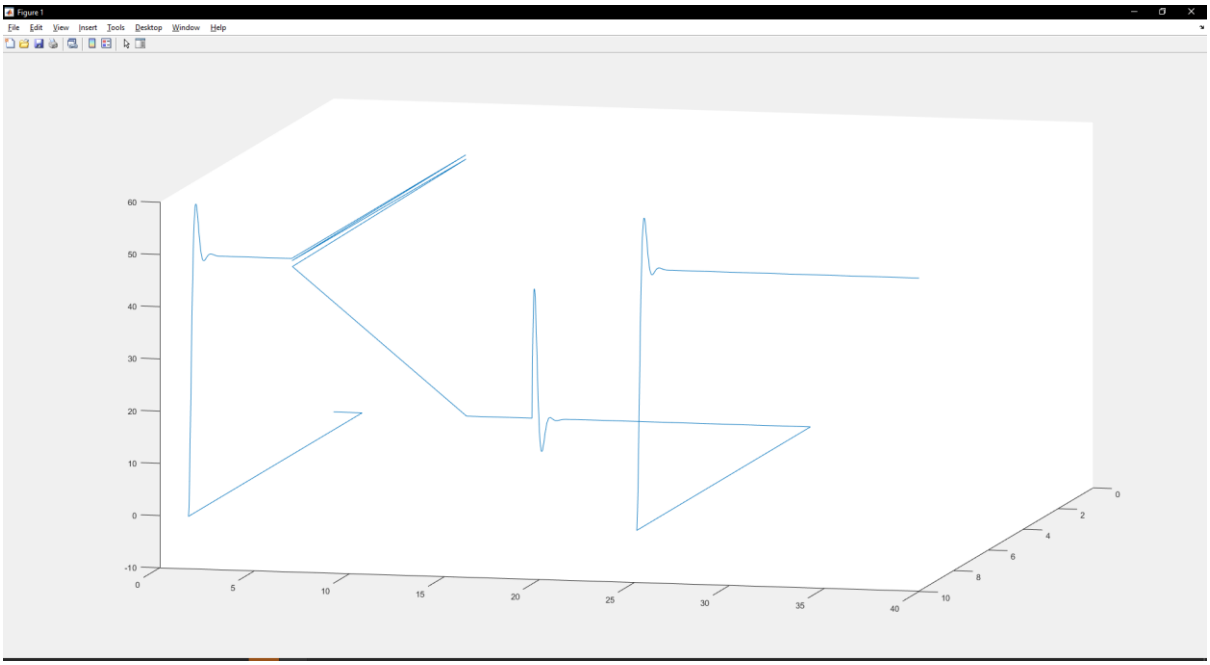

Simulation Data Inspector - Dataispector.mldatx
Inspect
Compare
Baseline: Run 16: AdvancedMechM34 (Sim
Compare to: Run 17: AdvancedMechM34 (SIL/
[+] More
Compare
16 Match
0 Mismatch
0 Not compared
Filter Comparisons
Properties
Chart1:6 (Run 16: AdvancedMechM34 (Simulation))
Chart1:6 (Run 17: AdvancedMechM34 (SIL/PIL))
Tolerance
Difference

Y PIL

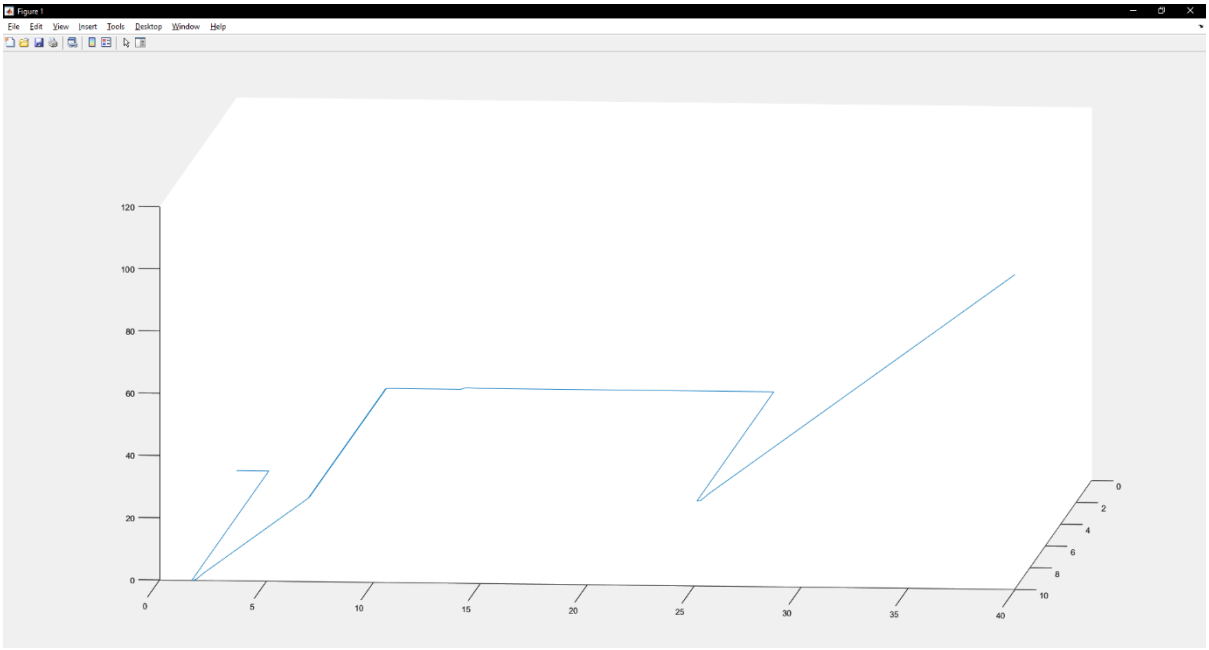
Hybrid State Trajectory:



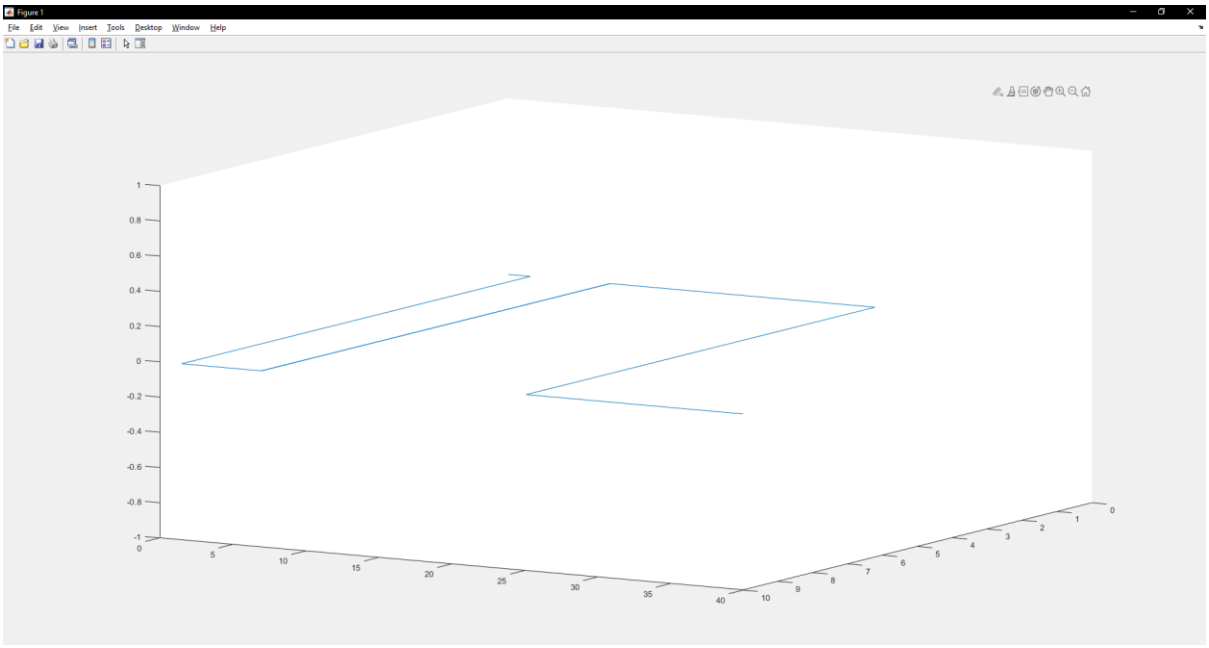
ThetadotA Hybrid state trajectory



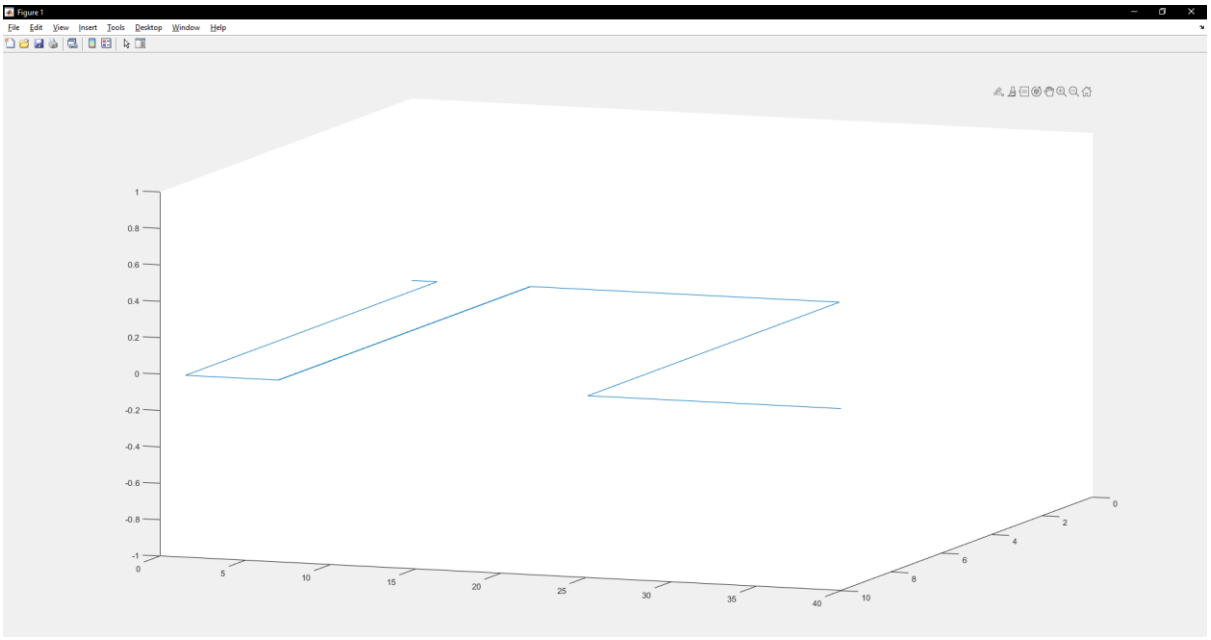
ThaetadotB Hybrid state trajectory



X Hybrid state trajectory



Y Hybrid state trajectory



Rotation Hybrid state trajectory