Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 13, Number 7 (2017), pp. 3121-3143

http://www.ripublication.com

Implementation of Real Time Operating System based 6-degree-of-freedom Missile Trajectory Simulation

Prathik Sargar
Symbiosis Institute of Technology, Lavale, Mulshi, Pune, Maharashtra, India

Dr. Himanshu Agrawal
Symbiosis Institute of Technology, Lavale, Mulshi, Pune, Maharashtra, India

Suman Mohandaas
Zeus Numerix Pvt. Ltd.
ISquareIT campus, Phase-1, hinjewadi, Pune, Maharashtra, India

Abstract

Simulation plays a key role in the development and testing of a missile system. Simulation not only helps to reduce research cost but also saves development time. Software in the loop simulation (SILS), Processor in the loop simulation (PILS), and Hardware in the loop simulation (HILS) are three ways by which simulation for the most of the system is achieved. SILS allows validation of model in precise fashion before field test. In SILS, modules of the system are replaced by executable code (for example missile's motion is represented by 6-DOF equation of motion) and ran on the PC platform. PILS is the intermediate stage between SILS and HILS. In PILS, the control algorithm will be run on a hardware platform to simulate a realistic situation. In HILS, modules like actuators, different sensors in simulation loop are actual hardware and others are simulated in more proficient and reasonable way. One of the key aspects of HILS is the precision of time and also it proves to be more economical. But for performing real-time HILS requires expensive platforms and hardware sensors. We are building low-cost simulation platform out of commodity equipment. We are carrying out PILS with the help Windows operating system. Windows OS has been a widely accepted operating system for desktop applications. However, it could not satisfy real-time aspect due to its uncertain system thread scheduling. So we are going to

use FreeRTOS along with it which is an open source real-time operating system that will meet real-time requirement of our system. We expect that the proposed technique can be used widely in the development of real-time applications where development cost and time are of key importance

Keywords: FreeRTOS, Software-in-the-loop, Processor-in-the-loop, Tracealyzer, Six-degree-of-freedom.

1. INTRODUCTION

Simulation of the real-time system has to be done before actual deployment of the system in its environment. Testing any critical system directly in its environment, not only increases development time but also its research cost. Also, some test conditions may not be always available in it real environment and can be simulated in the laboratory easily [7][11]. Complex missile system has to undergo intense testing before actual flight trials [9]. Simulation is the platform used for validating a mathematical model of the subsystems of the missile. SILS and HILS can prove the accuracy and usefulness of an aeronautical flight model [13]. The simulation can test the software and hardware design of missile and forecast the behavior of the system in actual flight test and will be able to find problems to improve the software design of missile.

Software in the loop simulation (SILS) is simulation based software evaluation. Simulated input conditions are given as input to the software system under evaluation to know how fine the system works under such input conditions. SILS is a cost-effective method for evaluating a complex, mission critical system before it can be deployed in real world environment. SILS is a useful tool to teach and understand better the concepts behind real aerial vehicles [3][11][12]. SILS and HILS are commonly used to evaluate the controls and algorithms because they help in the fast development and minimize experimental flights and finally reduce the overall cost required by the project [4][13]. The HILS systems are typically expensive and complicated compared to PILS. PILS provides a framework to verify the actual control algorithm on a dedicated microcontroller that controls plant simulation in the software environment [14]. PILS is much less expensive and simpler substitute for the assessment of the control algorithms. It introduces another level of simulation between SIL and HIL, with improved accuracy and closer approximation to the real plant and controller dynamics than SIL [14].

S.K.chaudhuri et.al[1] performed such HILS for the missile system. They highlighted guidance and control design issues, modeling and simulation techniques and validation methodologies for guided missile application. The flight software design along with on board computer, guidance, and control real-time hardware can be validated only in an integrated form in HILS. Similar such work was done recently using different platforms and real-time operating systems. Xiaofei Chang, et.al[6] have

done a study on results of digital simulation and hardware-in-the-loop simulation and have analyzed the deviation between the results for further study.

Chai Lina and Zhou Qiang built such HILS system based on RTX platform and windows environment [7]. RTX HAL extension is introduced in the Windows OS to overcome the poor real-time performance of the Windows OS. Qianlong Yang, et.al [8] also built HILS based on Windows operating system. They made use of external timer clock for their real-time requirements in order to improve timing accuracy and reliability simulated on Windows OS. Z.Y. Luo and M. Li[10] used rapid prototype method for building real-time simulation platform aiming at the requirements of the ballistic simulation. First mathematical model is built and the accuracy of the model and the reliability of the simulation platform is validated by HILS. They made use of VxWorks which is a commercial RTOS. It has advantages of high real-time capability, stable and reliable, scalable, widely used in the field of the missile, aircraft, etc. Also, there are another open source RTOSes like RT-Linux that can be used which have strong real-time features but, it can't be used in commercial projects unless we pay royalties and also platforms with small processor and microcontrollers can't support RT-Linux but something larger like x86 can support it.

There are many Real-Time operating systems available in the market. Some of the RTOSs available are VxWorks, RT-Linux, µs-OS, RTX for windows and eCos. The selection of the RTOS to implement the real-time model was mainly based on [16]:

(1) RTOS portability among different microprocessor/microcontrollers systems,

(2) Source code availability with ample documentation,

(3) Truly free for commercial applications and

(4) License to introduce alterations into the source code.

Moreover, available RTOSs are closed source and costly. We believe an open source RTOS with strong capability as that of commercial RTOS which can be used by any researcher who wants to perform real-time simulation in cost efficient way. As mentioned earlier, in a commercial project, the ability to use the GPL license doesn't always exist. In this case, it can be worth mentioning that FreeRTOS can be used in a commercial project without paying royalties. It is being ported to thirty-one different architecture and also has examples for different platform along with its source code. It only makes it easy to understand, learn and built the project. The development of the real-time system is fairly straight forward in freeRTOS. Since most of the operating systems are written in the c programming language, so are the real-time tasks themselves. So we selected FreeRTOS which is largely used RTOS.

In this paper we perform real-time simulation using Raspberry Pi 2 single-board chip as dedicated hardware and freeRTOS as the real-time operating system, for achieving low-cost SILS and PILS simulation platform. Also, we are not using any data acquisition sensor but we use digital sample values instead of analog data. We are building simulation platform out of commodity equipment so that hardware-based

simulation can possible with a low budget. Our contribution is summarised as follows:-

1) As per our knowledge this is the first attempt to use freeRTOS for real-time simulation for guided missile trajectory .
2) Use of low-cost equipment like raspberry pi to perform real time processor in the loop simulation so that we can check the validity of our control algorithm.

We are showing how the development of missile system can be done using open source tools like freeRTOS and development board like raspberry pi.

In the next section, we give preliminaries. In Sect. 3, we show our design and implementation flow. Sect. 4, gives details about the simulation results and the discussion. Finally, in Sect. 5 we will conclude and summarizes our findings.

2. MATHEMATICAL PRELIMINARIES:

In order to predict the trajectory of a guided missile, six degrees of freedom mathematical model is presented.

The equations of motions that describe the Six-DOF model are derived according to following assumptions:

a) The flying body is rigid.

b) All equations are referred to a body-fixed reference frame.

c) The aerodynamic coefficients are calculated in the body-fixed reference frame.

d) The Earth model is included (rotation, gravity).

A Six-DoF equation of motion model consists of three translational and three attitude degrees of freedom. The three translational equations of motion (linear velocity components: u, v, w) are derived from Newton's Law expressed in body coordinates and the three attitude equations of motion (angular velocity components: roll, pitch and yaw rates; p, q, r) are derived from Euler's Law expressed in body coordinates.

2.1 Mathematical Symbols:

Table 1: Nomenclature

p = Components of the angular velocity vector of the projectile in the body reference frame along x-axis.
q = Components of the angular velocity vector of the projectile in the body reference frame along y-axis.
r= Components of the angular velocity vector of the projectile in the body reference frame along z-axis.
u = Components of the velocity vector of the mass centre of the composite body in

the body reference frame along x-axis.

v = Components of the velocity vector of the mass centre of the composite body in the body reference frame along y-axis.

w = Components of the velocity vector of the mass centre of the composite body in the body reference frame along z-axis.

I_{xx} ,I_{yy} ,I_{zz} = Moment of inertia of the rocket about the X, Y, Z-axes

M = Mass of the missile.

V = Magnitude of the velocity vector of the mass centre of the projectile.

Φ, θ, ψ = Euler roll, pitch, and yaw angle of the projectile respectively.

R_e = Radius of the earth at the place of launch = 6356750 m

ω_e = angular velocity of the earth = 7.27×10-5 rad /s

λ = latitude at the place of launch

g = gravitational force.

2.2 Translational Equation of Motion: Newton's law:

The Six-DOF equations of motions are three translational degrees describe the motion of mass (Center of Gravity (CG)), also called the trajectory, as shown in the equation below.

Newton's second law with respect to the inertial frame "I" states that the time rate of change of linear momentum equals the externally applied forces. The external forces acting on the aerospace vehicles are aerodynamics and trust forces "$F_{a,p}$" and the gravitational acceleration.

$$mass \times acceleration = external\ forces$$

$$M\frac{dV}{dt} + V\frac{dM}{dt} = f = F_{external} \quad (Inertial\ Frame)$$

M $\frac{dV}{dt}$ + M[Ω]BE[u v w] = $F_{a,p}$+ [T]BLMg - 2[Ω]EI [u v w] - [Ω]EI[Ω]EISBI (body frame)

$$[T]^{BL} = \begin{bmatrix} \cos\psi\cos\theta & \sin\psi\cos\theta & -\sin\theta \\ \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \cos\theta\sin\phi \\ \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

Where,

$[T]^{BL}$ is the transformation matrix from local to body coordinate system.

[Ω]BE skew symmetric matrix of body rates p, q, and r.

[Ω]EI skew symmetric matrix of Earth's rotation

SBI Distance between Body C.G. and center of the Earth.

Scalar differential equation of the Newton, law

$$\dot{u} = \frac{F_x}{m} - qw + rv - (2\omega_e v \sin\lambda - T_{11}R_e\omega_e^2\sin\lambda\cos\lambda - T_{31}R_e\omega_e^2\cos^2\lambda) + T_{13}g$$

$$\dot{v} = \frac{F_y}{m} - ru + pw - (2(\omega_e \cos\lambda w + \omega_e \sin\lambda u) - T_{12}R_e\omega_e^2\sin\lambda\cos\lambda - T_{32}R_e\omega_e^2\cos^2\lambda) + T_{23}g$$

$$\dot{w} = \frac{F_z}{m} - pv + qu - (2\omega_e v \cos\lambda - T_{13}R_e\omega_e^2\sin\lambda\cos\lambda - T_{33}R_e\omega_e^2\cos^2\lambda) + T_{33}g$$

The integration of these differential equations yields the velocity vector that must be integrated once more to obtain the location of the rocket c.m. with respect to an earth reference point.

$$\begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix} = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\psi\sin\phi & \cos\theta\cos\phi \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

Where,

x_e, y_e, and z_e are the distance from the place of launch

Hence, there are six first-order differential equations that govern the translational motions of a vehicle with the earth as the inertial reference frame.

2.3 Rotational Equation of Motion: Newton's Law:

The rotational degrees of freedom are governed by Euler's law that states that the time rate of change of angular momentum equals the externally applied moments.

Time-rate-of-change of angular momentum = external moments

$$\omega\frac{dI}{dt} + I\frac{d\omega}{dt} = M_{a,p} \text{ (Inertial frame)}$$

$$I\frac{d\omega}{dt} + I[\Omega][p\ q\ r] = MB_{a,p} \text{ (body frame)}$$

$$\left[\frac{dp}{dt}\frac{dp}{dt}\frac{dr}{dt}\right] = [I]\text{-}1\ (-[I][\Omega][p\ q\ r] + M_{a,p})$$

Where,

$$[I] = \begin{bmatrix} Ixx & Ixy & Ixz \\ Iyx & Iyy & Iyz \\ Izx & Izy & Izz \end{bmatrix}$$

$$[\Omega] = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix}$$

The integration of these three differential equations yields the body rates that must be integrated once more (Euler angle differential equations) to obtain the orientations of the rocket c.m. w.r.t an earth reference point.

$\dot{\varphi} = p + q \tan\theta \sin\varphi + r \tan\theta \cos\Phi$

$\dot{\theta} = q \cos\varphi - r\sin\Phi$

$\dot{\psi} = q \sec\theta \sin\varphi + r \sec\theta \cos\Phi$

Hence, there are six first-order differential equations that govern the rotational motions of a vehicle with the earth as the inertial reference frame.

2.4 FreeRTOS:

FreeRTOS is a popular real-time operating system for embedded devices, being ported to thirty-one microcontrollers. It is distributed under the GPL(General Public License) with an optional exception. The exception permits user's trademarked code to remain closed source while maintaining the kernel itself as open-source, thereby aiding the use of FreeRTOS in proprietary applications. FreeRTOS is designed to be small and simple. The kernel itself consists of only three C files. To make the code readable, easy to port, and maintainable, it is written mostly in C, but there are a few assembly functions included where needed [13].

FreeRTOS is chosen because it's open source real-time operating system and development is easy using freeRTOS. It is easy to understand and learn and also it has a huge set of demo projects for different platforms provided for initial understanding and development of simple projects. All we need to do is make a project using the existing demo and include the libraries provided by freeRTOS. And it is mostly recommended to use the existing demo project and then make the changes according to our requirements. This is done so that some of the libraries are not missed while creating a new project and to avoid any issue while building the project.

2.5 Tracealyzer:

Tracealyzer gives exceptional understanding into the run-time world of RTOS-based embedded software, accelerating firmware development. RTOS-based development requires a good knowledge of the different timings (like execution time, waiting time,

response time, etc) of tasks, their interaction among each other, etc. Tracealyzer gives different graphical views into the run-time environment that helps during debugging, validation and optimization of the real-time system. It increases development efficiency and delivers robust, responsive real-time system [14].

3. DESIGN OF PROPOSED SYSTEM:

3.1 Architecture of 6-DOF Missile System:

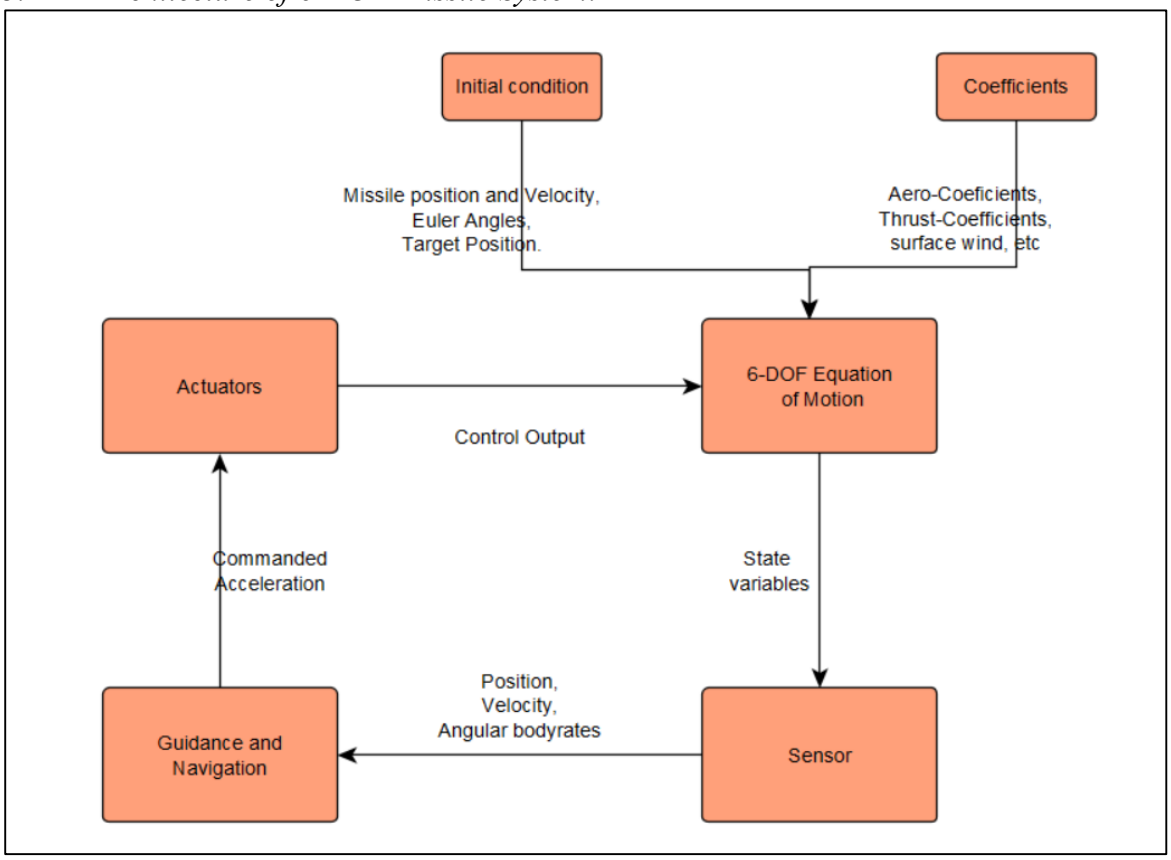

Figure 1: Architecture of 6-DOF Missile

Real-time simulation for the guided missile system started with building linear non-real-time (NRT) based model, evolving into a real-time (RT) based missile model. First, a mathematical model for the guided missile (i.e a Fire control computer model, six-degree-of-freedom mathematical model, and trajectory correction model) is built and simulation is set up by using Eclipse IDE. Then real-time simulation environment is set up using freeRTOS inside Eclipse IDE. FreeRTOS provide a port for Windows OS for simulating the real-time environment. The trajectory model is verified by real-time simulation. We are going test model using the real-time software in the loop simulation and compare the guided flight results with the results of the mathematical model. Finally, we will perform processor in the loop simulation using the raspberry pi board.

As mentioned earlier that we are only using the raspberry pi board as the hardware test bed and there are no actuators to control or no data acquisition sensor but instead digital signal being used. So there are only 2 tasks i.e Trajectory correction task and the Six-DOF task which will be running in parallel. Semaphores are used to take care of shared variables of the two tasks so that only one task will be manipulating the shared variables at a time. The Six-DOF task and trajectory correction are running with the same priority task.

In the beginning, flight control computer will do the simulation based on longitude, latitude and altitude values of source and the target. This simulation helps in finding out the elevation angle for firing the missile, total time the missile is going to take to hit the target, range value (distance between the source and the target) and bearing value. Elevation angle and bearing values are set and then we can carry out either free flight simulation or guided flight simulation depending on the requirement.

3.2 Simulation Parameters:

The main data and firing conditions of the studied guided missile are summarized below:

Table 2: Simulation parameters used in the simulation experiment

DATA	VALUE
Initial Center of Gravity C.G_xi	0.0 m
Final Center of Gravity C.G_xf	0.3975 m
Initial Axial Moment of Inertia. I_xxi	0.14 Kg.m2
Final Axial Moment of Inertia. I_xxf	0.12 Kg.m2
Initial Lateral Moment of Inertia. I_yyi = I_zzi	41 Kg.m2
Final Lateral Moment of Inertia. I_yyf = I_zzf	33 Kg.m2
Firing Elevation Angle (theta)	21 degree
Longitude of source and target	14600 and 0
Latitude of source and target	0 and 19800
Altitude of source and target	0 and 0

3.3 Source Code Snippet:

Table 3: Simulation source code used in the simulation experiment

```
int main( void )
{
        read_input();
        prvInitialiseHeap();
        vTraceInitTraceData();
        xTickTraceUserEvent = xTraceOpenLabel( "tick" );
        fflush( stdout );
        uiTraceStart();
        solver();
        return 0;
}
void solver()
{
        enableFlushAfterPrint();
        mutual_exclusion = xSemaphoreCreateMutex();
        intialize();
xTaskCreate(solver_6dof_main, (signed char*)"6-dof-solver", 1024, NULL, 1, &xHandle);
xTaskCreate(trajectory_correction,(signed char*)"trajectory_correction",1024, NULL, 1, &xHandle1);
        vTaskStartScheduler();
}

void solver(double var_6dof[10])
{
        //solving 6-dof equation of motion
}
void trajectory_correction(int *p)
{
        // Guidance, navigation, and control.
}
```

Execution begins with first instruction in main which calls to the read_input() (it is going to read all the coefficient required). prvInitialiseHeap() is related to memory usage. The project uses heap_5.c, so start by defining some heap regions. Next four lines are for trace record purpose (tracealyzer helps in generating statics report and different graphs for the two tasks). It will initialize the trace recorder and create the label used to post user events to trace recording on each tick interrupt. The solver is called next which initializes the required coefficients (i.e initializing aero_coefficents, thrust_data, surface wind, etc) and creates two tasks with xTaskCreate() API. This API allocates space in the heap for the task stack and it's Task Control Block (TCB). It then creates and initializes the various task queues that the kernel maintains, including the ready queue which is an array of FIFO queues, one for each priority and the delayed queue. It finally adds "6-dof-solver" to the ready queue and returns. Next, main calls xTaskCreate() for "trajectory_correction", which adds this task to the ready queue. Then, vTaskStartScheduler() API is called and it will creates the "idle" task with zero priority and adds it to the ready queue. The two tasks will run in parallel until the terminating conditions are met.

As microcontrollers usually have only one core, in reality only a single thread can be executing at any one time. The kernel chooses which thread should be executing by investigating the priority assigned to each thread. When one thread completes its execution, it will either be sent to completed queue or ready queue and another thread is chosen from the ready queue for execution and so on till the termination conditions are met.

3.4 Porting to Raspberry pi 2:

There is no official port for Raspberry pi2 board for using freeRTOS. But reference [18] provided a port for using freeRTOS with Raspberry pi. It provides port.c, portisr.c, and portmacro.h required by freeRTOS to run on specific hardware. We made use of this port so that the processor-in-the-loop simulation can be performed on the raspberry pi board.

Steps involved while running the bare metal program on raspberry pi are as follows:-

1. First, we have to copy three files to the SD-Card i.e bootcode.bin, start.elf, and kernel7.img.
2. Bootcode.bin will be first read by ROM. It will enable the SDRAM and load start.elf file which is bootloader.
3. Start.elf is GPU firmware. It is responsible for loading the other files and starts the CPU.
4. Kernel7.img is the bare metal application.

Loader.bin is an optional file. It is mainly used for loading start.elf file. bootcode.bin can also be used for loading start.elf file, if we add ".elf" loading support to bootcode.bin. Kernel7.img then executes the bare metal application on the pi board the way we programmed it to do. Config.txt and cmdline.txt are also optional files. Config.txt is used for setting system configuration just like bios does in a windows

environment. Cmdline.txt is responsible for passing arguments to the kernel (for example setting static IP address for raspberry pi).

In Linux environment, kernel7.img can be generated easily. We only need arm-none-eabi toolchain installed in Linux. If we want to generate kernel7.img file for raspberry pi on windows machine we have to make use of msys2 and the arm-none-eabi compiler or yagarto compiler for generating it. Kernel7.img is responsible for executing actions on the raspberry pi board. As mentioned earlier we do not use any dedicated data acquisition hardware. We transmit digital data sample instead of analog signals to the pi board so that we don't need any active components beyond the Raspberry Pi. And also we are not controlling any actuators, that is why we have only two tasks, one solves the Six-DOF equation of motion and the second does the guidance and navigation of the trajectory.

4. SIMULATION RESULT AND DISCUSSION

First, we conducted FCC (fire control computer) simulation. The basic parameters required to perform this simulation is longitude, latitude, and altitude values of source and the target. Once the FCC simulation is done we set up the firing parameters i.e setting elevation angle, and bearing angle. Setting up these angles is important so that missile can easily converge towards its target. Then we have performed mathematical simulation followed by real time SILS using freeRTOS and real time PILS using freeRTOS. While running in the non-real time-based environment there will be only one task or function calling other functions from within. So while doing mathematical simulation main function will call 6-DOF function and trajectory correction (guidance and navigation) function will be called from the 6-DOF function. The simulation will be completed within few seconds as it is run inside the Windows OS.

Now in real time simulation, we have created 2 tasks to emulate real time situation i.e 6-DOF task and trajectory correction task. These two tasks will be running in parallel and the shared variables between these 2 tasks are protected by semaphores so wrong results are avoided and the 2 tasks give correct output. The results of non-real time based mathematical simulation and the real-time SIL simulation are shown below with different graphs. The first graph gives altitude and range of the missile trajectory. The blue curve represents simple mathematical based solution and the other represents real time SILS solution. Both curves are almost same and converge towards the target. The little difference is caused due to the tasks created. In mathematical solution, there is only one task (one linear program) but in real time system, there are 2 tasks running in parallel and semaphore controlling access to shared variables. The second graph gives details on the drift of the missile trajectory and so on. From the graphs below it can be observed that almost all graphs are overlapping proving that real-time simulation carried out is indeed accurate.

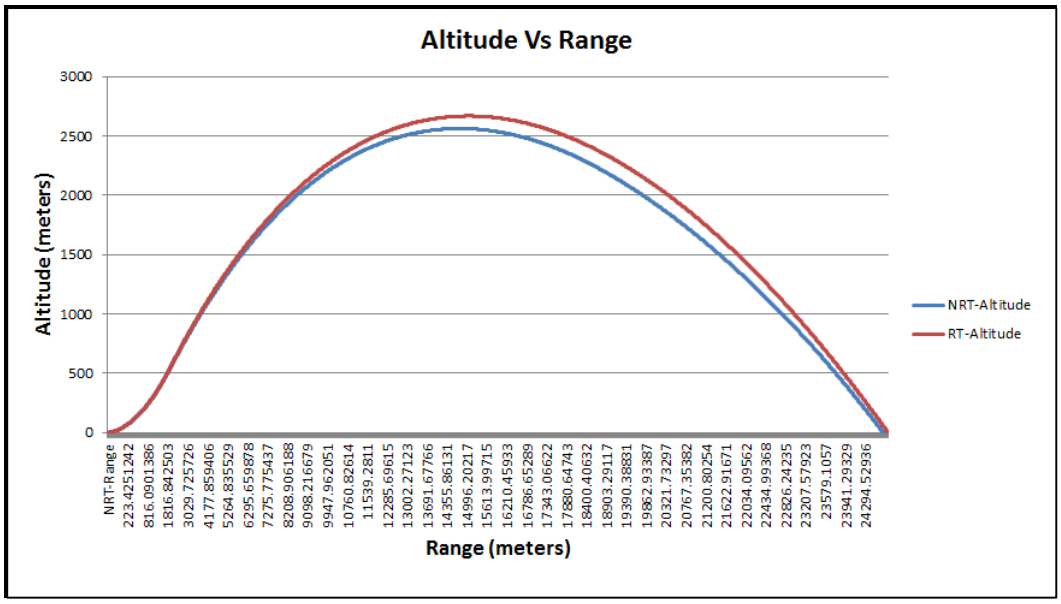

Figure 2: Range Vs Altitude

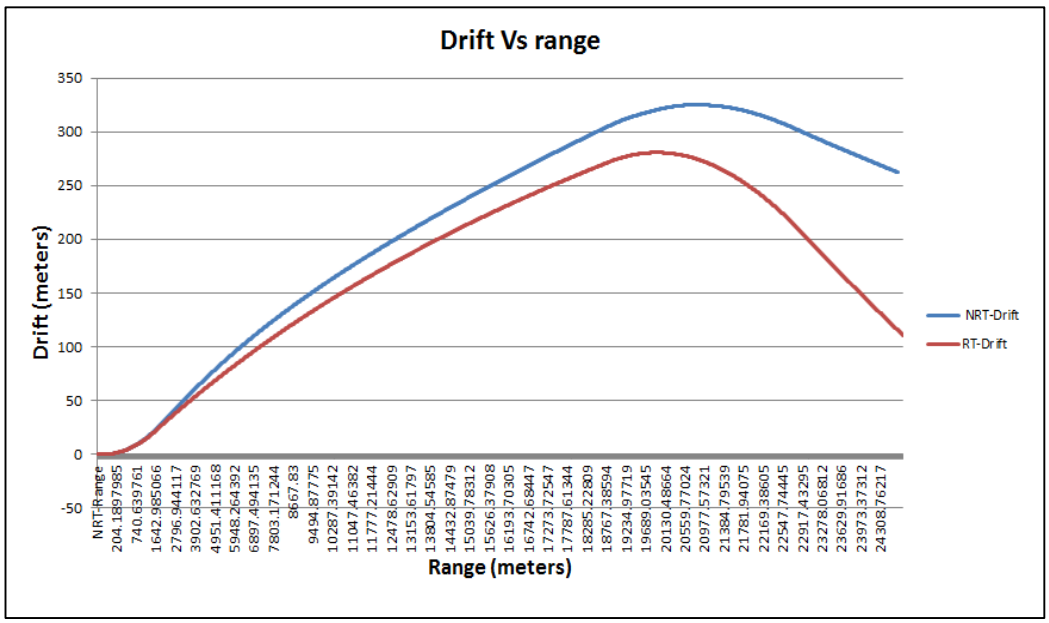

Figure 3: Range Vs Drift

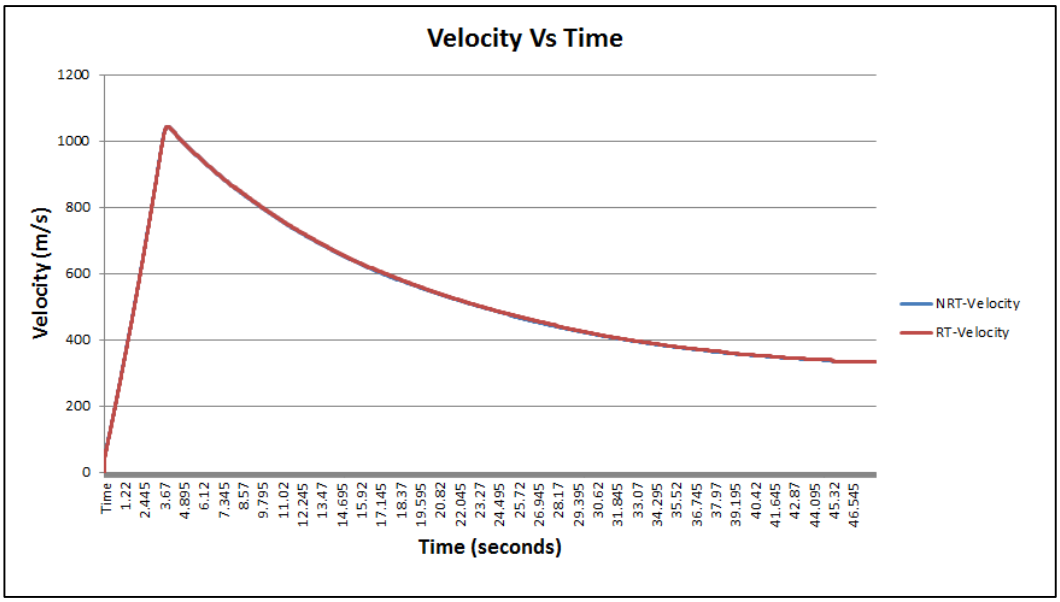

Figure 4: Velocity Vs Time

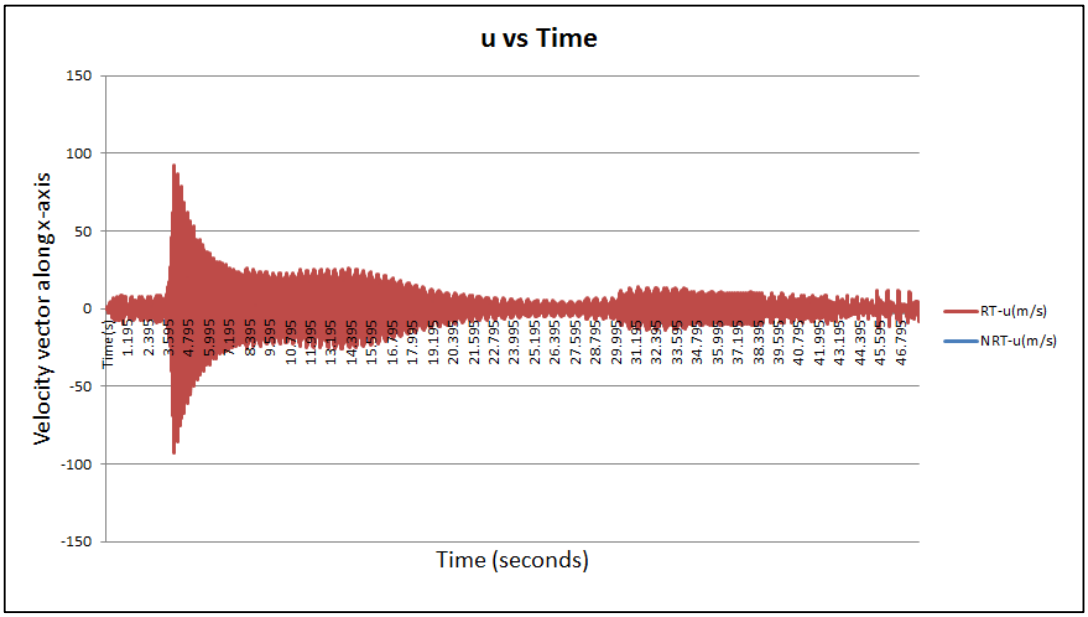

Figure 5: Velocity Vector along x-axis Vs. Time

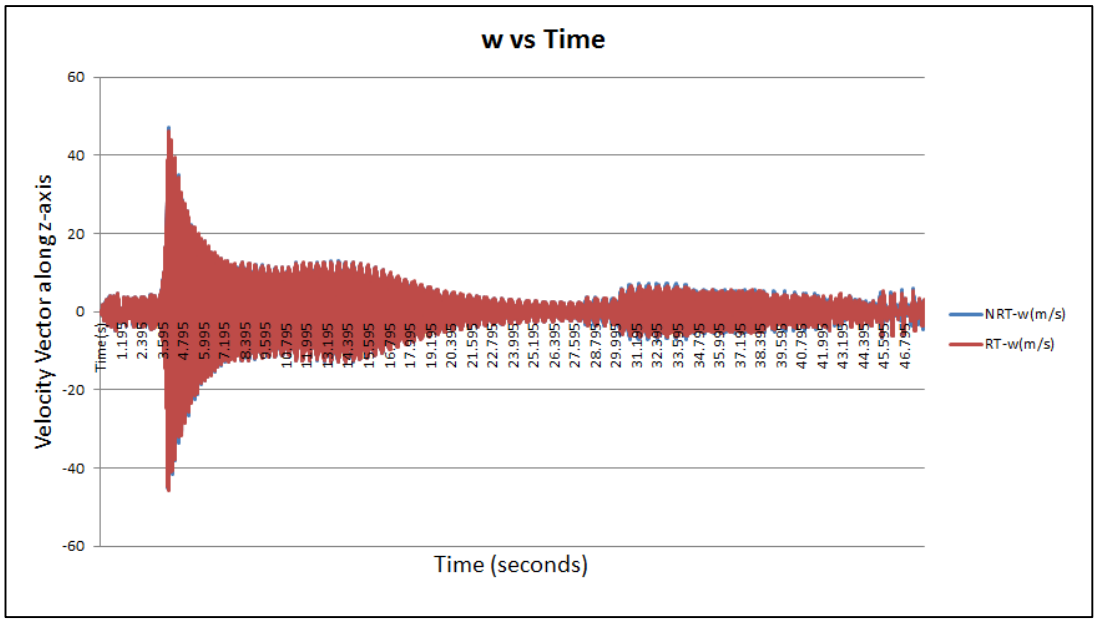

Figure 6: Velocity Vector along z-axis Vs Time

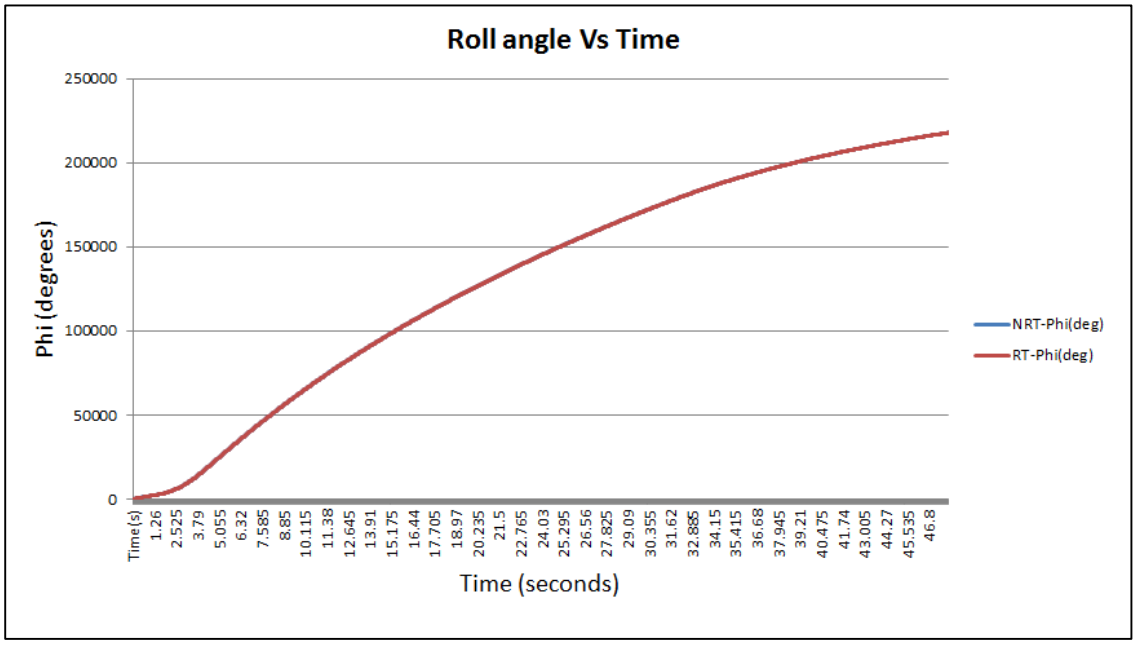

Figure 7: Roll Angle

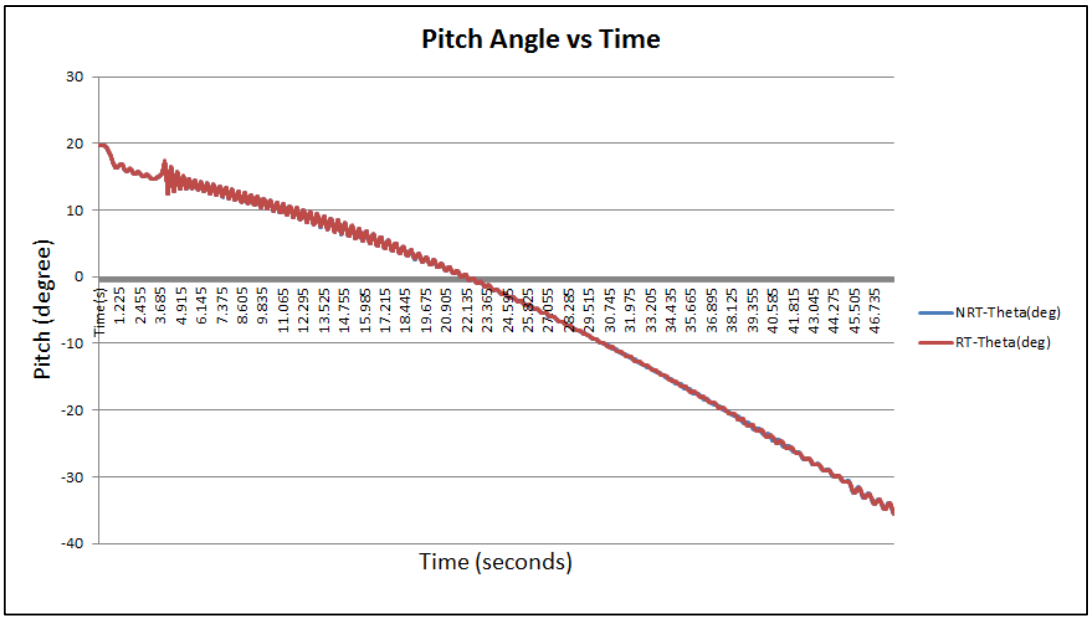

Figure 8: Pitch angle

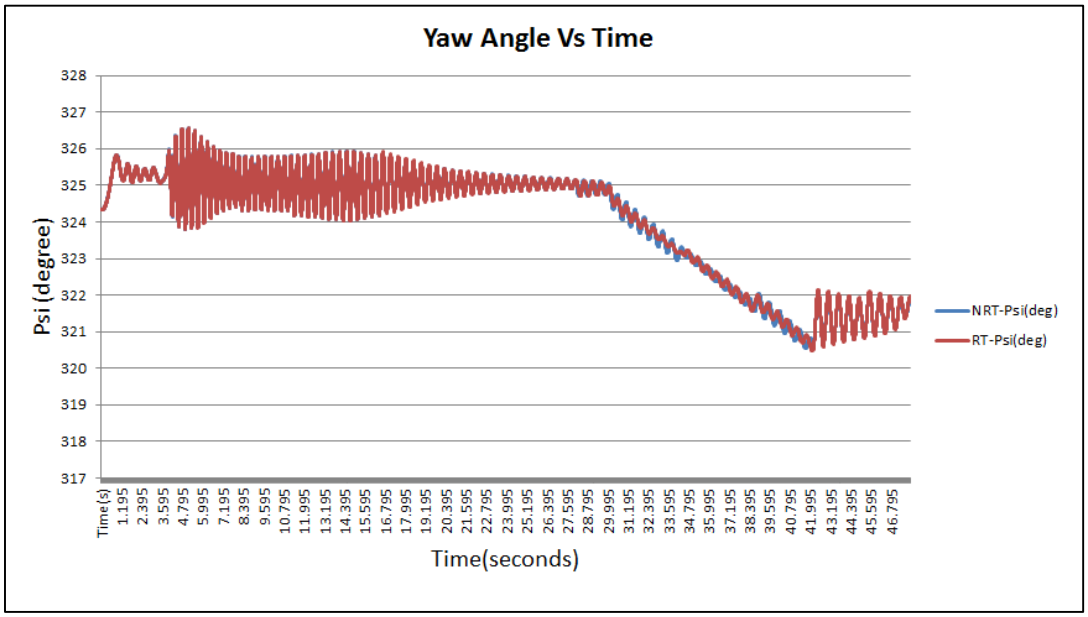

Figure 9 : Yaw Angle

Figure 10 and Figure 11 gives the 3-D plot of the missile. The first plot represents free flight missile trajectory. It means trajectory correction is not done here. It simply gives the path that missile is going to follow if no trajectory correction is applied. The second plot, on the other hand, represents missile's path when trajectory correction is

applied. As we can see from the figure that missile is changing its path to move towards the target.

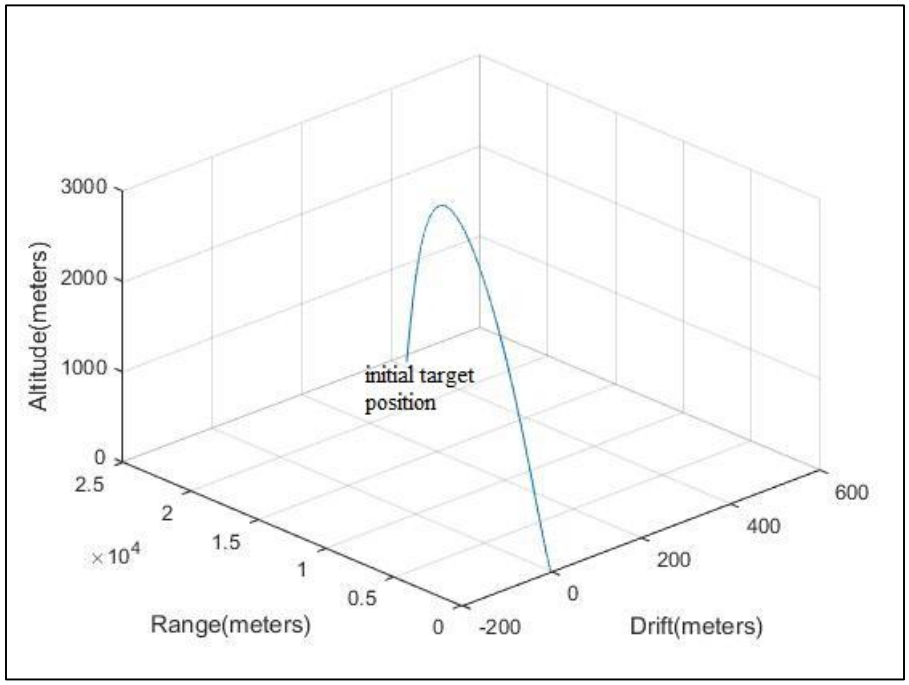

Figure 10: 3D plot of Free flight trajectory.

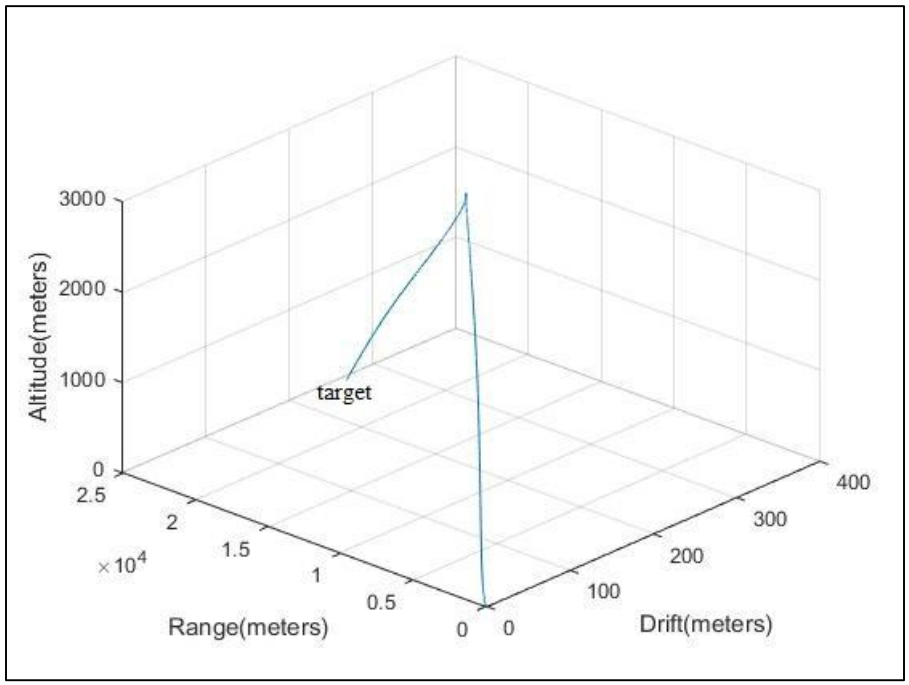

Figure 11: 3D plot of Guided flight trajectory.

We compared rotational angles (roll angle, pitch angle, and yaw angle), altitude, drift, and velocity vectors for validation of real-time simulation. Total time required to complete the simulation is 47.94 sec. The simulation cycle is of five milliseconds and so there are 9590 iterations. From the above graphs, we can see that results of real-time SILS matches with that of non-real time-based SILS proving that simulation carried out in real time is correct. As mentioned earlier Altitude values and Drift values are little different since in RT simulation there are actually 2 tasks running simulating a real time scenario, unlike the NRT simulation where there is only linear mathematical code being run. The solution file of the missile trajectory simulation is shown in figure 12,

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	Time(s)	Alpha(deg	Beta(deg)	Drift(m)	Height(m)	Mach	p(rpm)	Phi(deg)	Psi(deg)	q(deg/s)	r(deg/s)	Range(m)	Theta(deg	V	u(m/s)	v(m/s)	w(m/s)	Wind_dir(	wind_vel(m/s)	
2	0.25	-0.59366	-1.74357	0	0.1	0.137403	330	0	217	0	0	0	22.13	49	0	48.25023	0	69.97042	2.5	
3	0.255	-0.22863	-1.75857	3.61E-07	0.193288	0.140728	329.8821	9.898229	217	0.002144	-0.00812	0.2297	22.13001	50.16817	0.007987	49.4177	0.044947	69.97042	2.5	
4	0.26	0.119981	-1.70504	1.27E-06	0.288566	0.144054	329.7618	19.79287	216.9999	0.001026	-0.01719	0.464897	22.13003	51.3369	0.031185	50.58581	0.085798	69.97042	2.5	
5	0.265	0.440538	-1.58931	2.6E-06	0.385834	0.147382	329.6392	29.68385	216.9998	-0.00352	-0.02632	0.705594	22.13008	52.50617	0.068196	51.75456	0.118711	69.97042	2.5	
6	0.27	0.72308	-1.41952	4.29E-06	0.485095	0.150712	329.514	39.57109	216.9996	-0.01144	-0.03456	0.951795	22.13013	53.67598	0.116768	52.92394	0.140245	69.97042	2.5	
7	0.275	0.959552	-1.20518	6.32E-06	0.586348	0.154044	329.3862	49.45452	216.9994	-0.02243	-0.04096	1.2035	22.13021	54.84634	0.1739	54.09395	0.147535	69.97042	2.5	
8	0.28	1.143969	-0.95681	8.76E-06	0.689596	0.157378	329.2558	59.33406	216.9992	-0.03595	-0.04467	1.460713	22.1303	56.01724	0.23598	55.26457	0.138442	69.97042	2.5	
9	0.285	1.2725	-0.68557	1.17E-05	0.794839	0.160713	329.1231	69.20963	216.9989	-0.05127	-0.04494	1.723436	22.13041	57.18869	0.298945	56.43581	0.111677	69.97042	2.5	
10	0.29	1.34348	-0.40278	1.54E-05	0.902079	0.16405	328.9879	79.08116	216.9985	-0.06746	-0.0412	1.991672	22.13054	58.36068	0.358472	57.60767	0.066884	69.97042	2.5	
11	0.295	1.357354	-0.11961	1.98E-05	1.011317	0.167389	328.8505	88.94858	216.9981	-0.08348	-0.03311	2.265423	22.13069	59.53323	0.410179	58.78014	0.004677	69.97042	2.5	
12	0.3	1.316557	0.153357	2.53E-05	1.122554	0.17073	328.7109	98.81183	216.9976	-0.09819	-0.02054	2.544691	22.13085	60.70632	0.449839	59.95321	-0.07335	69.97042	2.5	
13	0.305	1.225344	0.406383	3.19E-05	1.235791	0.174072	328.5691	108.6708	216.997	-0.11043	-0.00366	2.829479	22.13104	61.87996	0.473582	61.12689	-0.16467	69.97042	2.5	
14	0.31	1.089555	0.630915	3.97E-05	1.351028	0.177416	328.4248	118.5255	216.9965	-0.11907	0.017113	3.11979	22.13124	63.05415	0.478089	62.30116	-0.26589	69.97042	2.5	
15	0.315	0.916358	0.819806	4.88E-05	1.468267	0.180762	328.278	128.3758	216.9958	-0.12309	0.041083	3.415627	22.13146	64.22888	0.460771	63.47603	-0.37291	69.97042	2.5	
16	0.32	0.713951	0.967496	5.92E-05	1.587509	0.184109	328.1288	138.2217	216.9951	-0.12162	0.067308	3.716991	22.13168	65.40415	0.419916	64.65149	-0.48102	69.97042	2.5	
17	0.325	0.491231	1.070145	7.07E-05	1.708755	0.187458	327.9774	148.063	216.9944	-0.11402	0.094633	4.023885	22.13191	66.57998	0.354804	65.82755	-0.58516	69.97042	2.5	
18	0.33	0.257464	1.125691	8.33E-05	1.832005	0.190809	327.8238	157.8997	216.9936	-0.09992	0.121735	4.336313	22.13215	67.75635	0.26579	67.0042	-0.68006	69.97042	2.5	
19	0.335	0.021953	1.133851	9.66E-05	1.957261	0.194161	327.668	167.7318	216.9928	-0.07922	0.147178	4.654276	22.13238	68.93327	0.154334	68.18145	-0.76049	69.97042	2.5	
20	0.34	-0.20629	1.096064	0.00011	2.084523	0.197515	327.5103	177.5591	216.9919	-0.05219	0.169475	4.977777	22.13261	70.11075	0.02299	69.35928	-0.8215	69.97042	2.5	
21	0.345	-0.41885	1.015382	0.000124	2.213793	0.200871	327.3503	187.3817	216.9909	-0.0194	0.18716	5.306819	22.13283	71.28877	-0.12465	70.53769	-0.85862	69.97042	2.5	
22	0.35	-0.60818	0.896302	0.000138	2.345072	0.204229	327.188	197.1994	216.9899	0.018204	0.198861	5.641404	22.13304	72.46734	-0.28405	71.71669	-0.86809	69.97042	2.5	
23	0.355	-0.76782	0.74456	0.000152	2.478361	0.207588	327.0234	207.0122	216.9888	0.059396	0.203377	5.981534	22.13324	73.64645	-0.44988	72.89626	-0.84704	69.97042	2.5	
24	0.36	-0.89261	0.56689	0.000166	2.613661	0.210949	326.8566	216.8199	216.9877	0.102656	0.199748	6.327213	22.13344	74.82611	-0.61618	74.07642	-0.79364	69.97042	2.5	
25	0.365	-0.97881	0.370751	0.000179	2.750974	0.214311	326.6876	226.6226	216.9864	0.146239	0.187311	6.678442	22.13362	76.00633	-0.77657	75.25716	-0.70724	69.97042	2.5	
26	0.37	-1.02418	0.164053	0.000191	2.890301	0.217675	326.5167	236.4202	216.9851	0.188241	0.165754	7.035223	22.13379	77.18709	-0.92451	76.43847	-0.58846	69.97042	2.5	
27	0.375	-1.02802	-0.04514	0.000203	3.031644	0.221041	326.3438	246.2126	216.9838	0.226672	0.135155	7.39756	22.13396	78.3684	-1.05352	77.62037	-0.43921	69.97042	2.5	
28	0.38	-0.99116	-0.24889	0.000214	3.175002	0.224408	326.169	255.9997	216.9823	0.25954	0.096006	7.765455	22.13411	79.55027	-1.15744	78.80283	-0.26268	69.97042	2.5	
29	0.385	-0.91585	-0.43965	0.000226	3.320379	0.227777	325.9922	265.7816	216.9808	0.284935	0.049219	8.13891	22.13426	80.73269	-1.23069	79.98587	-0.06329	69.97042	2.5	
30	0.39	-0.8057	-0.61054	0.000236	3.467774	0.231148	325.813	275.5581	216.9792	0.301128	-0.00389	8.517928	22.13439	81.91565	-1.2685	81.16948	0.153434	69.97042	2.5	
31	0.395	-0.66547	-0.75553	0.000247	3.617189	0.23452	325.6317	285.3291	216.9775	0.306655	-0.06161	8.902511	22.13451	83.09916	-1.26713	82.35366	0.380993	69.97042	2.5	
32	0.4	-0.50091	-0.86968	0.000257	3.768625	0.237894	325.4484	295.0947	216.9758	0.300308	-0.12193	9.292662	22.13461	84.28322	-1.22406	83.53842	0.612135	69.97042	2.5	

Figure 12: Solution file of 6-DOF missile

We made use of "Tracealyzer software" for debugging and optimizing our real time model. There is no delay while switching context between the tasks. We found out that context switching time involved is zero for real time SILS in a windows environment. Figure.13 shows how the 2 tasks are running and give all the execution details of the tasks involved. The first graph gives horizontal trace view of the tasks and the second graph gives details about CPU load graph of tasks. Horizontal trace view provides details such as start time, end time, execution time, response time, wait time, etc of each task instance.

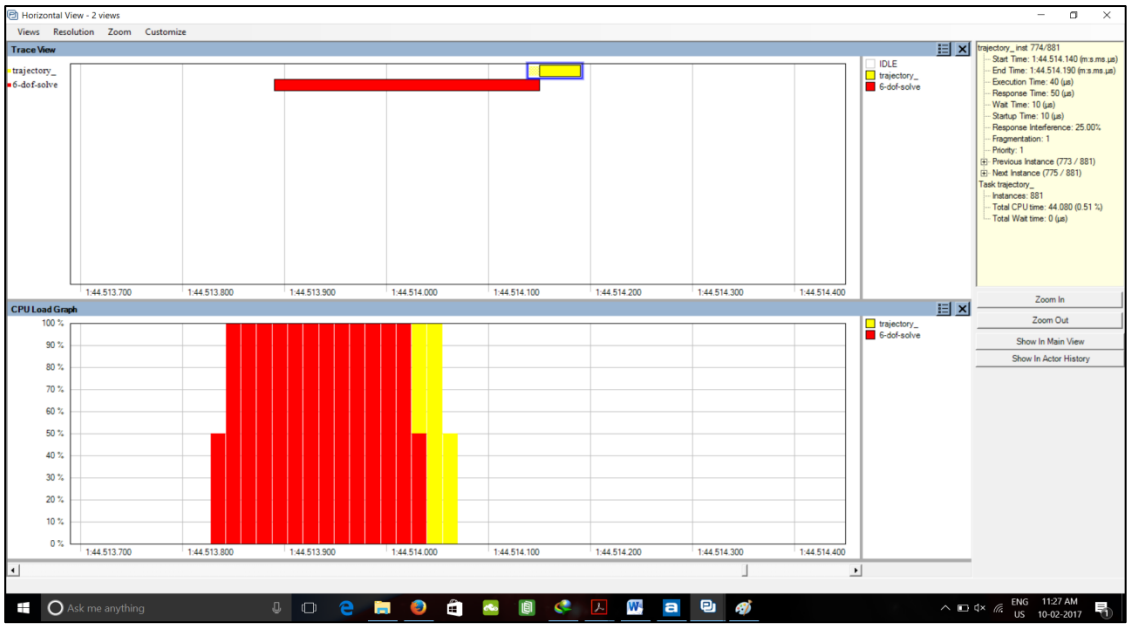

Figure 13: Trace view and CPU Load Graph

As we can see from the figure: 14, Six-DOF task and trajectory correction task runs periodically at the 5 os-tick gap (since simulation cycle i.e Δt is of 5 milliseconds). Figure 15 show the communication flow of the 2 tasks i.e the semaphore used for controlling access to critical section shared by the two tasks to avoid a race condition.

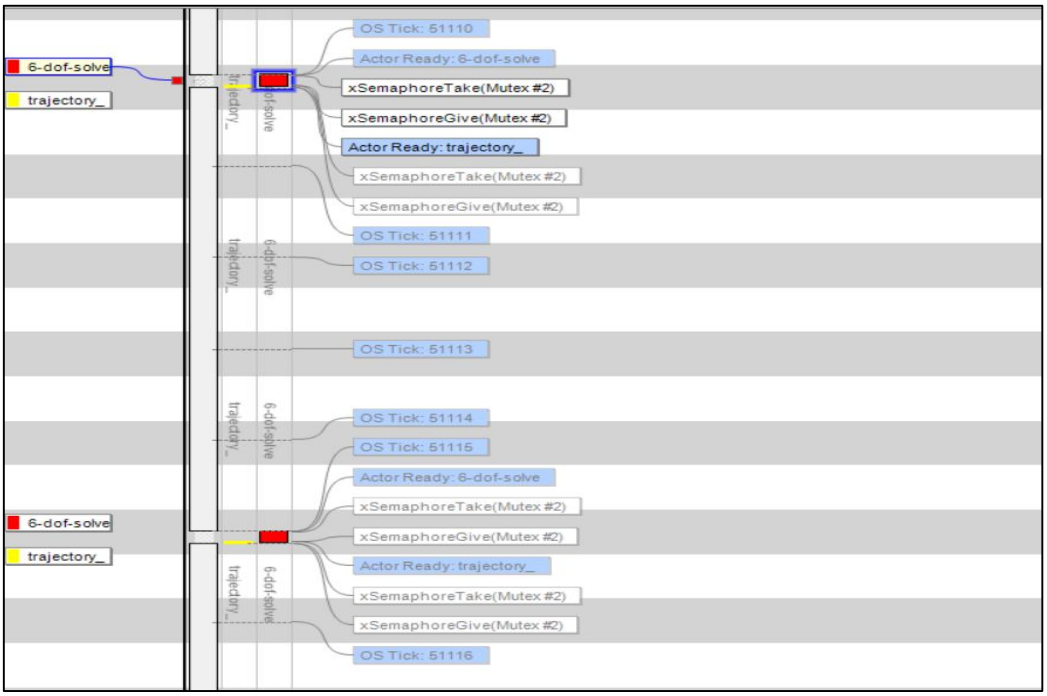

Figure 14: Trace Window

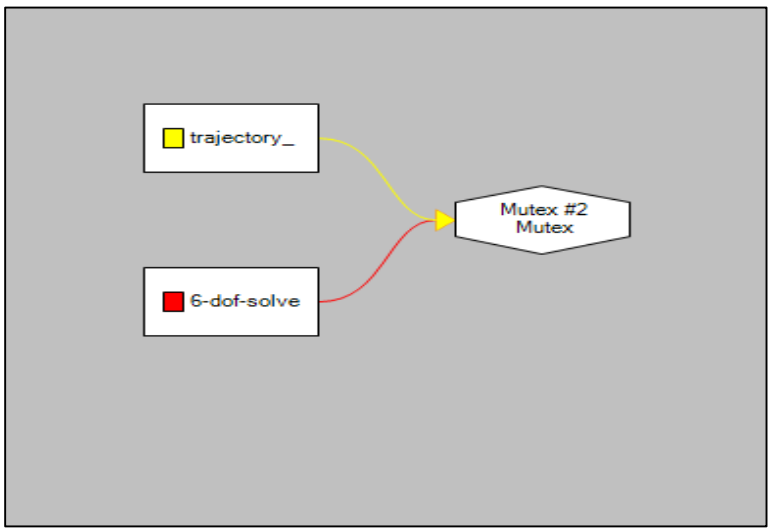

Figure15: Communication Flow

Figure 16 gives the statics report of the task:-

Included Actor Properties:

- **Priority:** The scheduling priority of the actor (Task, Thread or Exception).
- **Count:** The number of instances (jobs/executions) of the actor.
- **CPU Usage:** The amount of CPU time used by the actor (in percent).
- **Execution Time:** The actual CPU time used an actor instance (in microseconds).
- **Response Time:** The real time between start and completion an actor instance (in microseconds).
- **Periodicity:** The real time between the starts of two adjacent actor instances (in microseconds).
- **Separation:** The real time between the ends of an actor instance to the start of the next instance of that actor (in microseconds).

Note: The Min and Max values in the below table are links, which shows the corresponding actor instance in the trace window.

Actor	Priority		Count	CPU Usage	Execution Time			Response Time			Periodicity			Separation		
	Min	Max		%	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
Task trajectory_	1	1	882	0.551	30	50	340	40	60	430	7.400	9.710	12.430	7.330	9.650	12.370
Task 6-dof-solve	2	2	882	3.065	220	290	12.340	230	290	12.340	7.680	9.710	12.330	7.160	9.430	12.060

Figure 16: Statics report of two tasks.

Figure 17 gives actor instance graphs. The first graph represents periodicity from execution start of the instances of the 2 tasks. It represents the time between the instance start time and the previous instance finish. The second graph represents separation from execution start of the instances of the 2 tasks. It represents the time between the instance start time and the previous instance start time. Both separation and periodicity are available in two versions, depending on where to count the instance start time: from Ready and from Execution Start.

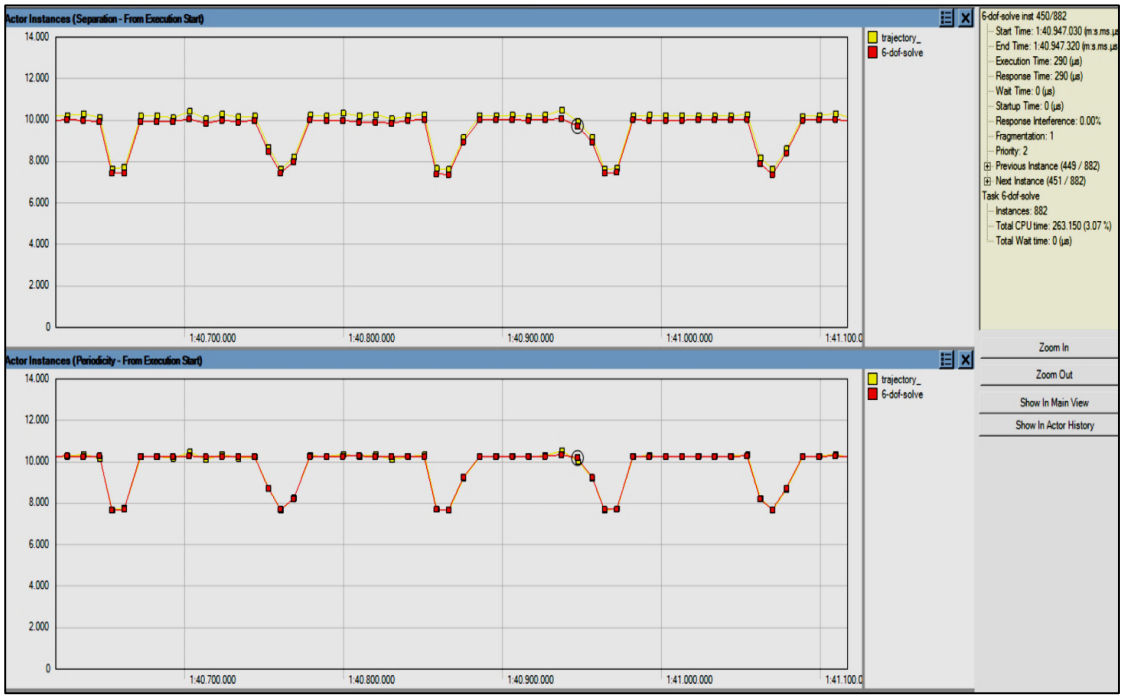

Figure 17: Separation and Periodicity from execution start of the task instances

5. CONCLUSION

We have introduced a new method to simulate Six-DOF missile trajectory using open source RTOS. We have shown how to build the real-time model for missile trajectory system. We made use of FreeRTOS running on top of windows operating system as a deployment platform for performing SILS. Finally, the results of non-real time-based SIL-simulation and real-time based SIL-simulation are compared for validation of the real-time model. Both the result matches proving that real-time simulation carried out using FreeRTOS is indeed valid. FreeRTOS switches context very fast in the win32 environment and hence can be used for the hard real-time system. At this moment we are able to run the tasks on the raspberry pi and see the output log on the monitor. As a future direction, we will be working on bare-metal FAT file-system driver for reading and writing data to either USB storage or sd card itself for storing output generated by the pi board instead of seeing it on the monitor. We are also currently working on "FreeRTOS + Trace" as there is no official trace library support for raspberry pi 2 yet for using tracealyzer for tracing the tasks on the raspberry pi.

6. ACKNOWLEDGEMENTS:

We express our deepest gratitude to Zeus Numerix Pvt. Ltd, Pune, India for providing the resources and support in our work. Our special thanks to Director of Zeus Numerix, Pune, Mr. Irshad khan. We express our deepest thanks to Dr.Sudhir Muthyala and Mr.Sanjay Kumar at Zeus Numerix, Pune, who took time out to hear, guide, and keep us on the correct path. Finally, we would also like to thank "Percepio AB" industry for providing a 1-year academic license for using Tracealyzer software.

REFERENCES

[1] S. K.chaudhuri, G. Venkatachalam and M. Prabhakar, "Hardware in the loop simulation for missile guidance and control", Research Centre Imarat, Hyderabad, Defence Science Journal-1997

[2] Siouris G.M, "The Generalized Missile Equations of Motion", Missile Guidance and Control System", Springer Journal-2004

[3] https://www.acm-sigsim-mskr.org/MSAreas/InTheLoop/softwareInTheLoop.htm

[4] S.K. Gupta, S. Saxena, Ankur Singhal and A.K. Ghosh, "Trajectory Correction Flight Control System using Pulsejet on an Artillery Rocket", IIT Kanpur. Defense Science Journal-2008.

[5] R.Liu, A.Monti, G. Francis, R. Burgos, F. Wang and D. Boroyevich", Implementing processor in the loop with universal controller in virtual testbed", IEEE, 2007.

[6] Xiaofei Chang, Tao Yang, Jie Yan, and Mingang Wang, "Design and Integration of Hardware-in-the-Loop Simulation System for Certain Missile", College of Astronautics, Northwestern Polytechnical University, Xi China, Springer Journal-2012

[7] Chai Lina, Zhou Qiang, "Design and Implementation of Real-Time HILS Based on RTX Platform", School of Automation Science and Electrical Engineering, Beijing University of Aeronautics and Astronautics Beijing 100083, IEEE, China

[8] Qianlong Yang, Qifeng Zhao, Tao Min, Chuanyan Tian, Gang Zhou, Fengqi Zhou , "Research of Flight Control Hardware-in-the-loop Simulation Based on Windows Operation System", People's Republic of China, IEEE-2013

[9] Rajesh S Karvande, B. Ramesh Kumar, "Development of Hardware-In-Loop Simulation Testbed for testing of Navigation System-INS", Research Centre Imarat, Hyderabad-500069, India, 2013

[10] Z.Y. Luo and M. Li, "Simulation environment for a missile based on Matlab or Simulink development", Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences, China, 2014

[11] Romulo Rodrigues, Rafael C. B. Sampaio, A. Pedro Aguiar, Marcelo Becker, "FVMS Software-in-the-Loop Flight Simulation Experiments: Guidance, Navigation and Control", Joint Conference on Robotics: SBR-LARS Robotics Symposium and Robocontrol, 2014.

[12] Adriano Bittar, Helosman V. Figuereido, Poliana Avelar Guimaraes and Alessandro Correa Mendes, "Guidance Software-in-The-Loop Simulation Using X-Plane and Simulink for UAVs.", International Conference on Unmanned Aircraft Systems (ICUAS), Orlando, FL, USA, 2014.
[13] Calvin Coopmans, Michal Podhradsk´y and Nathan V. Hoffer, "Software- and Hardware-in-the-Loop Verification of Flight Dynamics Model and Flight Control Simulation of a Fixed-Wing Unmanned Aerial Vehicle", Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS), Cancun, Mexico, 2015
[14] Harsh Vardhan, Bilal Akin, and Hua Jin, "A Low-Cost, High-Fidelity Processor-in-the-Loop Platform", IEEE power electronics magazine, June 2016.
[15] http://www.freertos.org/RTOS-task-states.html
[16] http://www.freertos.org/
[17] http://percepio.com/tz/freertostrace
[18] https://github.com/Forty-Tw0/RaspberryPi-FreeRTOS