JSST
Journal of Space Science and Technology

P-ISSN: 2008-4560
E-ISSN: 2423-4516

Journal of Space Science and Technology

Vol. 17, No. 2, pp. 81-91, 2024
https://doi.org/10.22034/jsst.2022.1351
Journal Homepage: https://www.jsstpub.com

Original Research Paper

Laboratory Development Approach in a 6DoF Launch Vehicle Simulation Design

Reza Esmaelzadeh[1*] and Fatemeh Zahra Rahmani[2]

1. Department of Aerospace Engineering, Malek Ashtar University of Technology, Tehran, Iran
2. Department of Aerospace Engineering, Amirkabir University of Technology, Tehran, Iran

ARTICLE INFO

Article History:

Received 15 June 2021
Revised 07 November 2021
Accepted 23 November 2021
Available Online 15 April 2022

Keywords:

6DoF simulation
Software in the loop
Hardware in the loop
Software engineering standards
RUP

ABSTRACT

A significant challenge in developing simulation software for flying objects is managing the transition from concept design to the final stages of hardware-in-the-loop integration. This paper introduces essential software engineering standards and procedures for developing robust, multi-stage launch vehicle simulation software using a novel approach to address this challenge. The proposed rational unified process (RUP) structure supports the rapid deployment of six degrees of freedom (6DoF) simulation software, allowing its application with minimal modifications in software-in-the-loop and hardware-in-the-loop laboratories. The paper discusses the standards and procedures for software production, followed by a detailed examination of the proposed simulation software structure. The RUP is recommended for developing 6DoF satellite simulation software, emphasizing that the programming expertise is more crucial than the choice of programming language. Given the Iranians strong programming expertise in C++, it is recommended as the programming language for 6DoF simulation due to its ease of debugging and faster development speed. Adhering to standard C++ ensures compatibility across C++ Builder, Turbo C++, and Visual C++ compilers with minimal modifications. Furthermore, the paper discusses the limitations of other languages, such as Fortran and Delphi, for subsystems like vehicle dynamics simulation (VDS), highlighting their weaker support for object-oriented programming. The conclusion supports the use of C++ for its robustness, flexibility across compilers, and strong development tools, thereby enhancing the efficiency and maintainability of satellite simulation projects.

*Corresponding Author's E-mail: esmaelzadeh@aut.ac.ir

How to Cite this Article:
R. Esmaelzadeh and F.Z. Rahmani, "Laboratory development approach in a 6DoF launch vehicle simulation design," *Journal of Space Science and Technology*, Vol. 17, No. 2, pp. 81-91, 2024, (in Persian), https://doi.org/10.22034/jsst.2022.1351.

JSST
علوم و فناوری فضایی

شاپای چاپی: ۴۵۶۰-۲۰۰۸
شاپای الکترونیکی: ۴۵۱۶-۲۴۲۳

علوم و فناوری فضایی

سال ۱۴۰۳، دوره ۱۷، شماره ۲، صفحه ۸۱-۹۱
https://doi.org/10.22034/jsst.2022.1351
Journal Homepage: https://www.jsstpub.com

مقاله پژوهشی

رویکرد توسعه آزمایشگاهی در طراحی ساختار کد شبیه‌سازی شش درجه آزادی ماهواره‌برها

رضا اسماعیل‌زاده [۱*] و فاطمه‌زهرا رحمانی [۲]

۱- دانشیار، مجتمع دانشگاهی هوافضا، دانشگاه صنعتی مالک اشتر، تهران، ایران
۲- کارشناسی ارشد، دانشکده مهندسی هوافضا، دانشگاه صنعتی امیرکبیر، تهران، ایران

اطلاعات مقاله

تاریخچه مقاله:

دریافت ۲۵ خرداد ۱۴۰۰
بازنگری ۱۶ آبان ۱۴۰۰
پذیرش ۰۲ آذر ۱۴۰۰
اولین انتشار ۲۶ فروردین ۱۴۰۱

واژه‌های کلیدی:

شبیه‌سازی شش درجه آزادی
آزمایشگاه نرم‌افزار در حلقه
آزمایشگاه سخت‌افزار در حلقه
استانداردهای مهندسی
RUP

چکیده

یکی از مشکلات جاری در توسعه نرم‌افزارهای شبیه‌سازی دینامیک، هدایت، کنترل و ناوبری اجسام پرنده، فرایند رشد و ارتقای نرم‌افزار از فاز طراحی مفهومی یک‌پروژه تا آخرین مراحل آزمایشگاه سخت‌افزار در حلقه است. در این مقاله، استانداردها و رویه‌های لازم مهندسی نرم‌افزار برای تولید نرم‌افزار شبیه‌سازی ماهواره‌برهای چندمرحله‌ای صلب چندمنظوره با رویکردی جدید معرفی می‌گردد تا بتواند بر این مشکل فائق آید. ساختار پیشنهادی RUP برای تولید نرم‌افزار شبیه‌سازی شش درجه آزادی، این قابلیت را به نرم‌افزار می‌دهد که به سرعت و با کمترین تغییرات در آزمایشگاه‌های نرم‌افزار در حلقه و سخت‌افزار در حلقه مورد استفاده قرار گیرد.

*پست الکترونیکی نویسنده مسئول: esmaelzadeh@aut.ac.ir

How to Cite this Article:
R. Esmaelzadeh and F.Z. Rahmani, "Laboratory development approach in a 6DoF launch vehicle simulation design," *Journal of Space Science and Technology*, Vol. 17, No. 2, pp. 81-91, 2024, (in Persian), https://doi.org/10.22034/jsst.2022.1351.

مقدمه

امروزه ابزار شبیه‌سازی در طیف وسیعی از کاربردها علمی و فنی مورد استفاده قرار می‌گیرد؛ توسعه الزامات عملکردی یک محصول، اعتبارسنجی طراحی، پشتیبانی از آزمایش‌ها، کاهش هزینه آزمایش‌ها، بررسی محیط‌های غیرقابل دسترس، آموزش کاربر، تمرین فرایندهای خطرناک، تحلیل دینامیک پرواز، تجمیع قطعات و تفریح، نمونه‌ای از این کاربردها هستند [۱].

شبیه‌سازی مسیر نقش مهمی را در توسعه ماهواره‌برها ایفا می‌کند. حجم زیاد مقالات و سایر متون علمی در این خصوص، مبین این اهمیت است. حین مراحل ابتدایی برنامه توسعه یک ماهواره‌بر، مهندسین برای ارزیابی طرح‌های جایگزین پیشرانش، پیکربندی، آیرودینامیک، محل پرتاب و محدودیت‌های آن، هدایت ، کنترل و ناوبری از شبیه‌سازی پرواز استفاده می کنند. شبیه‌سازی به دلیل همین مباحث چندگانه‌ای که در بر می‌گیرد می‌تواند چالش‌انگیز باشد. چارچوب مدل‌سازی و شبیه‌سازی ماهواره‌برها در مراجعی نظیر [۲-۴] مطرح شده است.

زبان‌های مورد استفاده در شبیه‌سازی را به دو دسته کلی می‌توان تقسیم کرد [۵]: زبان‌های دستوری یا Imperative هستند که در آن‌ها عبارات و الگوریتم‌ها در قدم‌های صریح تعریف می‌شوند؛ و در نقطه مقابل این نوع زبان‌ها، امروزه رویکرد نوینی از زبان‌های مدل‌سازی تحت عنوان زبان اعلانی یا Declarative نظیر Modelica [۶] توسعه یافته‌اند که در آن‌ها اعلانات از طریق معادلات انجام می‌شوند. در این مقاله صرفاً به زبان‌های دستوری پرداخته‌ایم.

به دلیل هزینه زیاد تست، اصلاحات زیاد و زمان طولانی تست واقعی سامانه‌ها، شبیه‌سازی سخت‌افزار درحلقه(HIL) که در آن بخش‌های نرم‌افزاری با سخت‌افزار جایگزین می‌شود روشی است که به خوبی برای اعتبارسنجی صحت الگوریتم‌های ناوبری [۷]، هدایت [۸] و کنترل [۹] تثبیت شده است. در این شبیه‌سازی عملکرد سخت‌افزارهای مختلف را در شرایط مختلف پروازی می‌توان بررسی نمود. امکان توسعه نرم‌افزار شبیه‌سازی به شبیه‌سازی HIL الزاماتی دارد که از ابتدای فرایند توسعه نرم افزار باید به آن‌ها توجه داشت. یکی از مشکلات جاری این حوزه فرایند رشد و ارتقای نرم‌افزار از فاز طراحی مفهومی یک پروژه تا آخرین مراحل آزمایشگاه HIL است که می‌تواند منجر به چالش‌هایی ناشی از تغییرات و اصلاحات فراوان و زمان‌بر شود.

اگرچه تولید نخستین نرم‌افزارهای شبیه‌سازی اجسام پرنده در کشور به اواسط دهه شصت باز می‌گردد و لیکن با وجود عمر نسبتاً کوتاه خود، تاکنون پیشرفت‌های قابل ملاحظه‌ای داشته‌اند. متاسفانه متخصصین هوافضایی عمدتا بر حسب تجربه اقدام به توسعه چنین نرم‌افزارهایی نموده‌اند و کمتر به استانداردهای توسعه نرم‌افزارهای هوافضایی توجه داشته‌اند. این مقاله الزامات و استانداردهای این حوزه را بررسی می‌کند.

ساختار این مقاله بدین‌صورت سازماندهی شده است که ابتدا به استانداردها و رویه‌های تولید نرم افزار پرداخته و سپس ساختار نرم‌افزار شبیه‌سازی پیشنهادی مورد بحث قرار داده می‌شود

استانداردها و رویه‌های تولید نرم‌افزار

با توجه به توسعه تجارت بین‌الملل و نیاز به استفاده از استانداردهای معتبر، مؤسسه بین‌المللی ISO با همکاری مؤسسه IEC[1] اقدام به تدوین استانداردهای بین‌المللی برای تولید و مستندسازی محصولات نرم‌افزاری نموده‌اند. استاندارد ISO/IEC12207 که در سال ۱۹۹۵ ارائه شد توصیه‌هایی برای کل چرخه ساخت و حیات یک محصول نرم‌افزاری پیشنهاد کرده است که به عنوان نمونه کاربردی به [۱۰] می‌توان مراجعه نمود. پس از آن IEEE به کمک مؤسسه EIA[2] اقدام به بومی‌سازی استاندارد ۱۲۲۰۷ در امریکا نمود و نسخه بومی‌شده و بهتر توصیف شده‌ای تحت‌عنوان IEEE/EIA12207 [۱۱] ارائه کرد. در نهایت DOD با پذیرش استاندارد مذکور، استانداردهای قبلی خود یعنی MIL-STD-498 و J-STD-016-1995 را از رده خارج کرد.

یکی از معتبرترین سازمان‌های استانداردسازی اروپا، استانداردهای آژانس فضایی اروپاست. اولین نسخه این استانداردها در سال ۱۹۸۴ منتشر شد که اکنون رعایت این استانداردها برای کلیه نرم‌افزارهای آژانس اجباری است. مرجع [۱۲] به تجزیه و تحلیل مفاد استاندارد ESA-PSS-05-00 نسبت به استاندارد ISO/IEC12207 پرداخته است. براساس این استاندارد مراحلی که باید در چرخه حیات یک نرم‌افزار طی شود عبارتند از:

- تعیین نیازهای کاربر و نرم‌افزار
- طراحی معماری نرم‌افزار و طراحی تفصیلی و تولید نرم‌افزار
- انتقال و واگذاری نرم‌افزار برای بهره‌برداری
- نگهداری و بهره‌برداری

فرآیند مهندسی نرم‌افزار مجموعه‌ای از قدم‌های قابل پیش‌بینی برای توسعه نرم‌افزار را مشخص می‌کند. پیروی از یک رویه منظم تولید نرم‌افزار به تولیدکنندگان نرم‌افزار کمک می‌کند امور مربوط به تولید نرم‌افزار را منظم و پروژه را در حداقل زمان ممکن و با کارایی بالایی انجام دهند. برای طراحی یک رویه تولید

1. International Electrotechnical Commission

2. Electronic Industris Association

نرم‌افزار می‌توان از روش‌های متفاوتی استفاده کرد و از آن جا که هر پروژه نرم‌افزاری با دیگر پروژه‌ها متفاوت است، می‌توان گفت رویه تولید آن پروژه نیز با دیگر پروژه‌ها تفاوت دارد. در واقع انتخاب این روش‌ها رابطه مستقیمی با اندازه گروه در پروژه دارد و نرم‌افزارهای بزرگ و کوچک نیاز به رویه‌های تولید متفاوت دارند. از مهم‌ترین این روش‌ها می‌توان به مدل‌های آبشاری، Scrum و RUP اشاره کرد.

مدل آبشاری که گاهی«چرخه حیات کلاسیک» یا «مدل ترتیب خطی» نامیده می‌شود، بیانگر نگرش نظام‌مند و زنجیره‌ای نسبت به تولید نرم‌افزار است که در سطح سیستم شروع شده و با تحلیل، طراحی، کدنویسی، آزمون و پشتیبانی نرم‌افزاری پیشروی می‌کند. DOD_STD_2167A به‌طورجامع در تولید نرم‌افزار بر مدل آبشاری تکیه دارد [۱۳]. در این روش سیستم به قسمت‌های کوچک تقسیم شده و سطوح مختلفی از این تقسیم‌بندی به وجود می‌آید. سپس برای نمایش و مدل‌سازی چگونگی گردش اطلاعات دیاگرام جریان داده (DFD)[1] ایجاد می‌شود. با پرداختن به جزئیات هر DFD می‌توان به‌تعریف فرایند رسید و در این مرحله یا کد مستقیماً نوشته می‌شود یا ابتدا طراحی الگوریتم انجام شده، سپس کد نوشته می‌شود. برای نمایش و مدل‌سازی ساختار اطلاعاتی ERD[2] ترسیم و از روی آن پایگاه داده طراحی می‌شود.

امروزه، یکی از روش‌های تولید نرم‌افزار که به‌خصوص برای پروژه‌های نرم‌افزاری کوچک مورد استفاده قرار می‌گیرد و توسط بسیاری از صاحب‌نظران مورد تأیید قرار گرفته‌است، روش Scrum است. با استفاده از این روش تکرارشونده، می‌توان نرم‌افزارهای بزرگ را به قسمت‌های کوچکتر تقسیم و سپس هر قسمت را با کیفیت بالا تهیه کرد. در این روش، مدیریت قوی تولید نرم‌افزار وجود دارد که به برنامه‌نویسان اجازه می‌دهد با استفاده از آن در پروژه‌ها به سرعت نرم‌افزار مورد نظر را تهیه نمایند. در این روش هر عضو از گروه موظف به درک وظیفه خود در پروژه است و باید یک هدف مشخص را در تمامی مراحل عملیاتی یا فازهای اجرایی دنبال کند.

اما چه تفاوتی بین Scrum و دیگر روش‌های تولید نرم‌افزار وجود دارد؟ در جواب این سؤال باید یادآور شد که در Scrum هر مرحله قسمتی از نرم‌افزار را آماده می‌کند. در این روش می‌توان پیشرفت در تولید نرم‌افزار را در هر مرحله به‌خوبی احساس کرد. روش Scrum وقتی می‌تواند بیشتر مفید باشد که در ابتدای پروژه نیازهای کاربران به‌صورت دقیق مشخص نباشد و یک گروه کوچک مسئول پروژه نرم‌افزاری باشد. مزایای استفاده از Scrum بسیار است اما این روش چند اشکال نیز دارد:

۱- روش جدیدی است و با روش‌های مرسوم تفاوت‌های زیادی دارد.
۲- برخی از برنامه‌نویسان حرفه‌ای ممکن است از تکالیفی که مدیر Scrum به ایشان می‌دهد راضی نباشند و بخواهند روش قدیمی خود را اجرا نمایند و در صورت اجبار، در روند اجرای پروژه کارشکنی کرده و مشکل‌آفرینی کنند.
۳- از آنجاکه مدیر Scrum هم از نظر کیفی و هم کمی باید پروژه را مدیریت کند، Scrum نیاز به مدیر بسیار قدرتمند دارد.
۴- Scrum را می‌توان به عنوان روش تولید نرم‌افزار نام برد، اما این روش بیشتر روش مدیریت پروژه هوشمند خوبی است و نمی‌توان آن را به‌صورت منفرد استفاده نمود. Scrum را از آن جهت می‌توان روش خوبی برشمرد که روشی تحقیقی براساس تخمین، اولویت‌بندی، عملکرد گروه و بررسی نتایج است که اگر به‌صورت صحیح مورد استفاده قرار گیرد و قبل از استفاده به‌صورت کامل آموزش داده شود، می‌تواند راندمان پروژه‌های نرم‌افزاری را به خصوص تولید نرم‌افزارهای زیرمجموعه که خود نیز در نهایت تشکیل نرم‌افزار بزرگی را خواهند داد، به‌صورت بسیار محسوسی بالا ببرد.

روش RUP[3] یکی از معروف‌ترین رویه‌های تولید نرم‌افزار بوده که توسط شرکت IBM طراحی شده است. در حقیقت هدف اصلی RUP اطمینان از این موضوع مهم است که آیا نرم‌افزار تولیدشده نیازهای کاربرانش را به‌صورت کامل، با کیفیت بالا، در زمان معین و با بودجه مشخص برآورده کرده است یا خیر. RUP دارای سه جزء اصلی است: جزء اول از مجموع راه‌حل‌های خوب که در رویه می‌تواند مورد استفاده قرار گیرد تشکیل شده است. جزء دوم همان مراحل تهیه نرم‌افزار است و جزء آخر قسمت‌های تشکیل‌دهنده این رویه است. RUP شش راه‌حل خوب را که می‌تواند در مراحل مختلف این رویه به ما کمک کند معرفی کرده است [۱۴]:

۱- استفاده از Use-Case ها که می‌توانند در جمع‌آوری نیازهای کاربران مفید باشند،
۲- استفاده از معماری نرم‌افزار قابل استفاده مجدد،
۳- استفاده از روش‌های تکرارشونده برای کنترل بهتر و آسان پروژه،
۴- استفاده از نمودارهای UML،
۵- کنترل تغییرات در نرم‌افزار،
۶- کنترل کیفیت نرم‌افزار با توجه به درخواست‌های اولیه کاربران

تولید نرم‌افزار شبیه‌سازی مبتنی بر فرضیات و اصولی است که لازم است به آن‌ها توجه نمود [۱۵]:

1. Data Flow Diagram
2. Entity Rlationship Diagram
3. Rational Unified Process

- هیچ شبیه‌سازی منفرد و یکپارچه‌ای نیاز همه کاربران را برآورده نمی‌کند؛ کاربران دارای علائق و نیازهای متفاوتی در دقت و جزئیات هستند،
- توسعه‌دهندگان شبیه‌سازی دارای دانش متفاوتی در موضوع مورد شبیه‌سازی هستند،
- هیچ‌کس نمی‌تواند انتظار داشته باشد همه کاربردهای شبیه‌سازی ترکیب شوند؛ دنیا در حال تغییر است و هیچ‌کس کاربردهای آتی شبیه‌سازی را حتی در یک حوزه نمی‌داند.
- باید امکان استفاده از فناوری و ابزارهای آتی لحاظ شود.

با این فرضیات، توسعه دهندگان شبیه‌سازی باید اهداف ذیل را درنظر داشته باشند [۱۵]:

- باید امکان تجزیه یک مسئله شبیه‌سازی بزرگ به بخش‌های کوچکتر وجود داشته باشد، بخش‌های کوچکتر را به راحتی و صحیح‌تر می‌توان تعریف، ایجاد و اعتبارسنجی کرد،
- باید امکان ترکیب بخش‌های کوچک‌تر نرم‌افزاری به شبیه‌سازی بزرگ وجود داشته باشد،
- باید امکان ترکیب این بخش‌های کوچک به سایر بخش‌های حتی پیش‌بینی نشده برای ایجاد یک شبیه‌سازی جدید وجود داشته باشد،
- توابع عمومی باید از شبیه‌سازی خاص جدا شوند، در مورد استانداردهای معماری نرم‌افزار که لازم است در شبیه‌سازی رعایت شوند مرجع [۱۵] نکات کابردی قابل توجهی مطرح نموده که در ساختار پیشنهادی نرم‌افزار شبیه‌سازی که در بخش بعد ارائه می‌شود مطرح شده است.

ساختار پیشنهادی نرم‌افزار شبیه‌سازی

زیرسیستم‌های نرم‌افزار شبیه‌سازی را از دید کاربردی به دو نوع می‌توان تقسیم کرد: زیرسیستم برنامه پرواز [1](OFP) یا به‌طورکلی زیرسیستم‌هایی که حین پرواز استفاده می‌شوند و زیرسیستم‌های دینامیکی که حین پرواز استفاده نمی‌شوند که خود دو قسم است:

الف– نرم‌افزارهایی که هیچ ارتباطی با OFP ندارند نظیر شبیه‌ساز دینامیک ماهواره‌بر VDS [2]

ب – نرم‌افزارهایی مرتبط با OFP

این تقسیم‌بندی به خاطر اهمیت OFP و برنامه‌های مرتبط با آن است. برنامه OFP مثل سخت‌افزارهای سیستم هدایت و کنترل، بالاترین کلاس یعنی کلاس نظامی دارد. بنابراین از خصوصیات برنامه OFP، قابلیت اطمینان بالا و حساسیت آن است. برنامه OFP یک ماهواره‌بر می‌تواند حداقل شامل ماژول‌های زیر باشد:

- ماژول سیستم کنترل ماهواره‌بر
- ماژول سیستم هدایت
- ماژول سیستم ناوبری
- ماژول پردازش سیگنال‌های سنسورهای کمک ناوبری
- ماژول سیگنال‌های تله‌متری (در صورت نیاز)
- ماژول مدیریت فرامین سیستمی‌ماهواره‌بر
- ماژول سیستم تشخیص شکست جهت انهدام خودکار

یکی از نرم‌افزارهایی که هیچ ارتباطی با OFP ندارند، زیرسیستم VDS است. این زیرسیستم حداقل شامل ماژول‌های زیر است:

- ماژول حل معادلات دیفرانسیل
- ماژول حل محاسبات تانسوری، برداری و ماتریسی
- ماژول خواندن ورودی‌های پیشرانش و آیرودینامیک
- ماژول مدل اتمسفر
- ماژول مدل جاذبه زمین
- ماژول معادلات ماهواره‌بر
- ماژول درون‌یابی و برونیابی عددی
- ماژول محاسبات تراست و جرمی– اینرسی ماهواره‌بر
- ماژول شبیه‌ساز IMU
- ماژول شبیه‌ساز سیستم سرومکانیزم
- ماژول شبیه‌ساز تأخیرهای سخت‌افزاری و پسخور فرامین مدیریت سیستمی OFP

یکی دیگر از نرم‌افزارهایی که هیچ ارتباطی با OFP ندارند، زیرسیستم مقداردهی اولیه پارامترهای VDS است. این زیرسیستم وظیفه‌اش این است که فایل یا پارامترهای نیروی پیشران، فایل یا ضرایب آیرودینامیکی، پارامترهای جرمی، اینرسی و ابعادی ماهواره‌بر، فایل خطاهای پارامترهای محیطی، جرمی، اینرسی و انرژتیک، ترجیحات شبیه‌سازی مثل چند مرحله‌ای بودن را از کاربر دریافت دارد. همچنین این زیرسیستم ممکن است شامل ویرایشگر طراحی موتور ماهواره‌بر که خروجی آن فایل پیشرانش و پارامترهای موتور است و ویرایشگر کد ضرایب آیرودینامیکی به‌صورت کد MD یا Mark4 باشد.

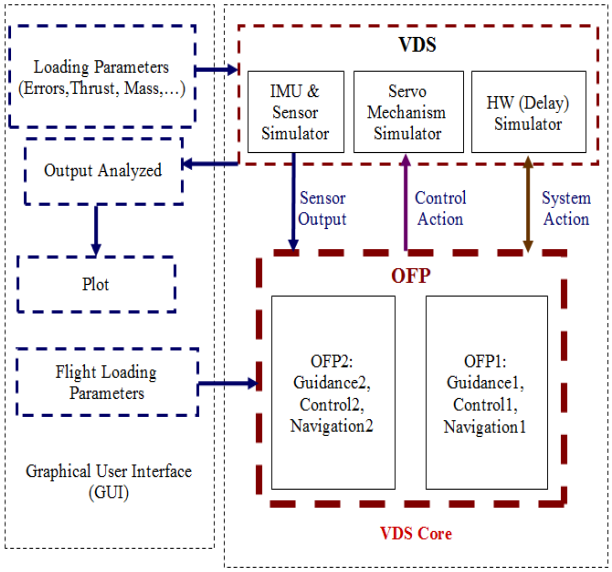
زیرسیستم آنالیز خروجی، زیرسیستمی است که وظیفه آن تهیه محیط گرافیکی کاربرپسند برای تحلیل پارامترهای VDS و OFP است.

1. Onboard Flight Program
2. Vehicle Dynamic Simulator

این زیرسیستم همچنین می‌تواند به‌صورت لحظه‌ای از خروجی‌های VDS و OFP، مدل خطی شده دینامیک ماهواره‌بر را در اختیار کاربر قرار دهد.

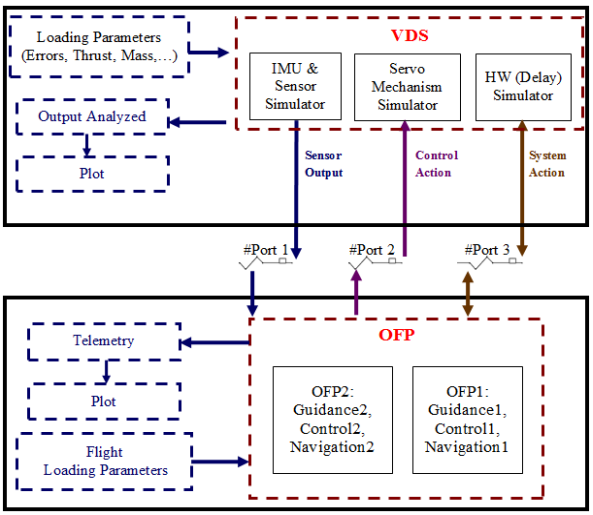
زیرسیستم بارگذاری پارامترهای پرواز، جزء نرم‌افزارهای مرتبط با OFP است. این زیرسیستم محیطی گرافیکی کاربرپسند برای ورودی‌های OFP یا VDS فراهم می‌کند. همچنین می‌تواند امکاناتی نظیر محیط گرافیکی طراحی مسیر ماهواره‌بر در اختیار قرار دهد. شکل (۱) ارتباط این زیرسیستم‌ها را نشان می‌دهد. همان‌طورکه پیداست، مزایای این معماری برای شبیه‌سازی ماهواره‌بر شامل موارد ذیل است:

- قابلیت حمل برنامه پرواز، به عبارتی به راحتی قسمت برنامه پرواز از سایر زیرسیستم‌ها قابل تشخیص و جداسازی است،
- کاهش مراحل میانی توسعه آزمایشگاه‌های SIL و HIL،
- با مدل‌سازی خوب تأخیرهای سخت‌افزاری، سطوح کوانتیزاسیون دیجیتال A/D، نویز سرومکانیزم و همچنین IMU می‌توان از تعداد تست‌های طراحی الگوریتم‌های هدایت و کنترل در آزمایشگاه HIL به طور چشمگیری کم کرد.
- در زمان شبیه‌سازی مونت کارلو، بسته به روش هدایتی انتخاب شده با جداسازی هسته شبیه‌سازی از زیرسیستم‌های مقداردهی اولیه صرفه‌جویی شده زیرا در شبیه‌سازی‌های تکراری فقط یکبار بارگذاری و مقداردهی اولیه OFP و VDS انجام می‌پذیرد.



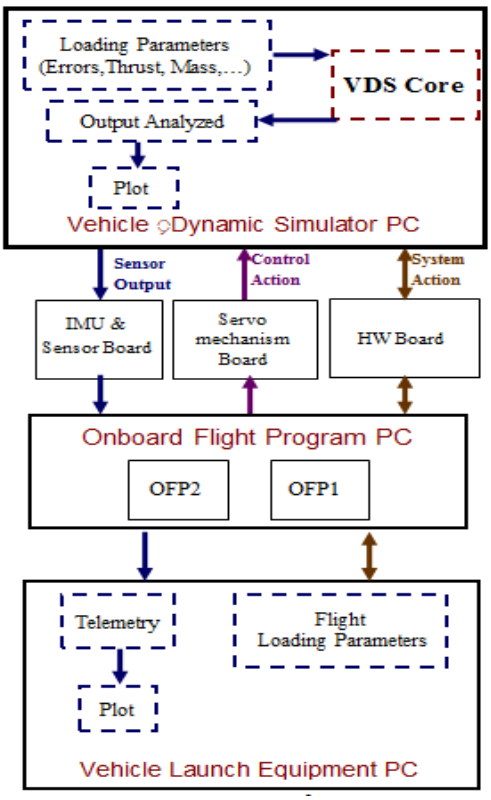
شکل ۱– زیرسیستم‌های برنامه شبیه‌سازی شش درجه آزادی.

Fig. 1. Subsystems of the Six Degree of Freedom simulation scheme.



شکل ۲– اولین قدم ایجاد آزمایشگاه نرم‌افزار در حلقه ساده.

Fig. 2. The first step to create the software lab in a simple loop.



شکل ۳– آزمایشگاه نرم‌افزار در حلقه نهایی.

Fig. 3. Software lab in the final loop.

قابلیت آزمایشگاه نرم‌افزار در حلقه

با ساختار شبیه‌سازی شش درجه آزادی پیشنهادی و با دوتکه‌کردن آن به‌صورت شکل (۲)، یک شبیه‌سازی نرم‌افزار در حلقه ساده ایجاد

می‌گردد. این شبیه‌سازی شامل دو برنامه OFP همراه با برنامه‌های مرتبط با آن در یک رایانه و برنامه VDS همراه با برنامه‌های مرتبط، در رایانه دیگر است. این دو برنامه می‌توانند توسط رابط‌های خاص با هم تبادل اطلاعات نمایند. برای انتقال داده، در بسیاری از کاربردهای هوافضایی از پروتکل‌هایی مانند MIL-1553، ARNIC-429 و RS422 استفاده می‌شود. اما اگر بر فرض از نرم‌افزار Simulink استفاده شود انتقال داده‌ها توسط کارت‌های سخت‌افزاری که شرکت MathWorks در اختیار کاربران قرار داده و با استفاده از پروتکل‌های TCP/IP یا RS232 انجام می‌شود. همچنین می‌توان این دو برنامه را با تعریف رابط‌های مجازی روی یک PC نیز اجرا نمود.

در یک رایانه برنامه هدایت، کنترل و ناوبری ماهواره‌بر قرار می‌گیرد و در رایانه دیگر معادلات شبیه‌ساز دینامیک و IMU جای می‌گیرد. این دو رایانه با پورت‌های سریال، شبکه یا چاپگر با یکدیگر ارتباط برقرار می‌کنند. مزایای طراحی شبیه‌سازی نرم‌افزار در حلقه عبارتند از:

۱) بررسی صحت و اعتبار جداسازی برنامه هدایت، کنترل و ناوبری از برنامه دینامیک ماهواره‌بر،

۲) قابل حمل بودن آزمایشگاه،

۳) تست صحت و زمان برقراری پورت‌های مربوطه،

۴) معماری و تولید ارتباطات I/O در برنامه .OFP

آزمایشگاه نرم‌افزار در حلقه را می‌توان قدم‌به‌قدم توسعه داد و به جای برخی از زیرسیستم‌های شبیه‌ساز سخت‌افزار از کارت‌ها یا بردهای الکترونیکی شبیه‌ساز سخت‌افزاری استفاده نمود. همچنین می‌توان زیرسیستم بارگذاری پارامترهای پرواز را نیز جدا کرد. رایانه سوم رایانه تجهیزات پرتاب (VLEPC)[1] می‌نامیم. شکل (۳) آزمایشگاه نرم‌افزار در حلقه نهایی را نشان می‌دهد.

قابلیت آزمایشگاه سخت‌افزار در حلقه

پس از طراحی الگوریتم‌های هدایت، کنترل و ناوبری ماهواره‌بر لازم است الگوریتم‌های هدایت و کنترل در شرایط شبه واقعی پیاده‌سازی گردند تا نقص‌های احتمالی طراحی و سازگاری سخت‌افزار با نرم‌افزار آشکار شود. جهت تشخیص نقص‌ها، خطاها و تلرانس نقص طراحی OFP، یک سیستم تشخیص نقص لازم است. سیستمی که توانایی‌اش آشکارسازی شکست و تشخیص است. ویژگی مهم در طراحی سیستم تشخیص نقص، طراحی قابلیت تست سیستم است که دارای دو خصیصه کنترل پذیری و رویت پذیری باشد [۱۶]. در فاز تست طراحی تفصیلی سیستم OFP، بسیاری از تست‌های OFP در آزمایشگاه سخت‌افزار در حلقه انجام می‌شود. در واقع، آزمایشگاه سخت‌افزار در حلقه یک سیستم تشخیص نقص برای OFP فراهم کرده است. این آزمایشگاه جهت تشخیص نقص‌های زیر کاربرد دارد:

- عدم سازگاری سخت‌افزار سرومکانیزم با اتوپایلوت
- عدم سازگاری زیرسیستم‌های یکپارچه شده قبل از پرواز
- عدم کارکرد مناسب سیستم در شرایط بحرانی و سخت
- نقایص‌عملکردی ناشی از الگوریتم‌های هدایت، کنترل و ناوبری
- عدم تطابق داده‌های تله‌متری نسبت به‌شبیه‌سازی6DOF
- نقص‌های ناشی از کارکرد سخت‌افزار هدایت و کنترل (چک جعبه رله‌ها، بارابلوک‌ها، دمای IMU، و...)
- نقص‌های ناشی از زمانبندی هدایت و کنترل یا مدیریت سیگنال‌ها (زمان رفت و برگشت سیگنال‌های جدایش، روشن شدن موتورهای مراحل و...) و کدینگ.

همان‌طورکه از شکل (۴) مشخص است، این آزمایشگاه دارای‌سه وظیفه است: آشکارسازی، شناسایی و برآورد شکست. وظیفه آشکارسازی شکست تصمیمی ساده است: آیا چیزی اشتباه است یا همه چیزعالی‌است؟ اگر لازم شد، قدم بعدی وظیفه شناسایی شکست است. مسئله شناسایی شکست (مجزاکردن یا تشخیص) تعیین منبع شکست است. اغلب تعیین منبع شکست به سادگی میسر نیست وگاهی برای تشخیص منبع شکست چند تست دیگر نیز تعریف می‌گردد. وظیفه نهایی ارزیابی شکست است که چگونه روی سیستم OFP تاثیر گذارده است. آشکارسازی شکست OFP در آزمایشگاه سخت‌افزار درحلقه یا به طور خودکار انجام می‌شود (با فعال شدن سیگنال‌های انهدام خودکار) یا با مشاهده پارامترهای رویت‌پذیر (پارامترهای تله‌متری) توسط اپراتور انجام می‌گیرد.

نرم‌افزار OFP دارای دو روش تست (تست‌های قبل از پرتاب یا تست‌های عملکرد سخت‌افزار) و مد پرواز است. تست‌های سخت‌افزاری، نرم‌افزاری و مد پرواز را می‌توان در قالب فرامینی تعریف کرد. فرمان‌ها برای اجرای رایانه OFP از طریق رایانه VLEPC ارسال می‌گردد. در مد تست (تست‌های قبل از پرتاب) بلادرنگ بودن برنامه‌های OFP حداکثر Soft Real Time است. ولی در مد پروازی برنامه OFP از نوع Hard Real Time بوده و محدودیت‌های زمانی و سنکرونیزاسیون

1. Vehicle Launch Equipment PC

دارد. برنامه در مد پرواز در فرکانس مشخص اجرا می‌شود، کل زمان فرآیند خواندن و نوشتن از روی A/D، D/A، شمارنده پالس‌های واحد اندازه‌گیر اینرسی، DIO و تله‌متری به علاوه زمان اجرای محاسبات الگوریتم هدایت، کنترل، ناوبری و سیستم تشخیص شکست حداکثر باید در حدود ۹۰٪ زمان واقعی باشد. ۱۰٪ زمان واقعی برای حصول قابلیت اطمینان در نظر گرفته می‌شود. در آزمایشگاه شبیه‌سازی HIL (به طور مختصر آزمایشگاه) اجزای اصلی زیر وجود دارند:

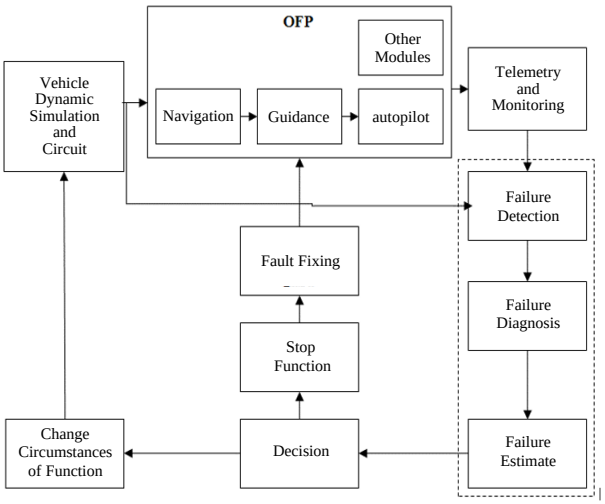
- نرم‌افزار OFP که روی یک رایانه پرواز قرار می‌گیرد،
- سرومکانیزم‌ها و مبدل‌های D/A و A/D،
- شبیه‌ساز دینامیک ماهواره‌بر

برخی اجزای دیگر مثل تله‌متری، رسم اطلاعات، میز چند درجه آزادی و ارتباطات اجزا فرعی آزمایشگاه هستند. آزمایشگاه سخت‌افزار در حلقه را می‌توان در دو مدل ساخت:

۱. شبیه‌سازی سخت‌افزار در حلقه مدل آزمایشگاهی (شکل ۵)

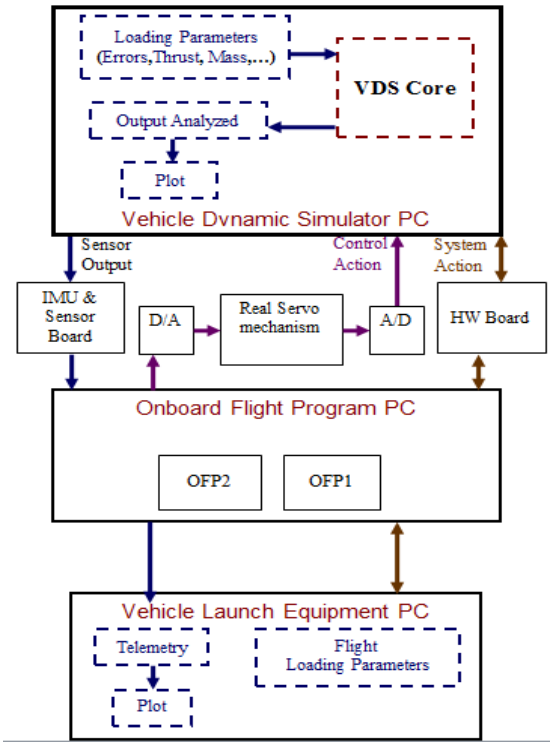
۲. شبیه‌ساز سخت‌افزار در حلقه مدل صنعتی (شکل ۶)

تفاوت عمده این دو مدل این است که در مدل صنعتی(عملیاتی) حداکثر سخت‌افزارهای ماهواره‌بر واقعی در اندازه‌های واقعی استفاده شده ولی در مدل آزمایشگاهی تنها از سرومکانیزم واقعی و برد الکترونیکی شبیه‌ساز جدایش‌ها و روشن‌شدن موتورها استفاده می‌شود. در مدل عملیاتی در برخی از تست‌ها از IMU واقعی استفاده خواهد شد درصورتی‌که در مدل آزمایشگاهی چنین امکانی فراهم نخواهد شد.



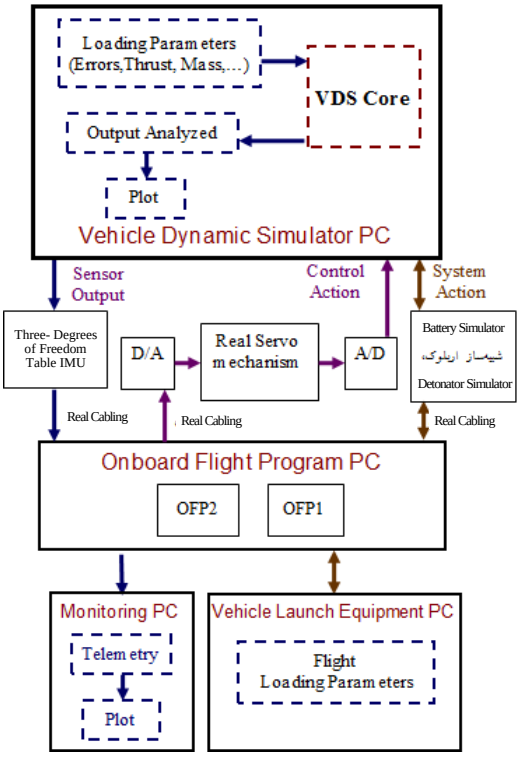
شکل ۴- سیستم تشخیص نقص OFP در آزمایشگاه HIL.

Fig. 4. OFP fault diagnosis system in the HIL laboratory.



شکل ۵- بلوک نمودار مدل آزمایشگاهی سخت‌افزار در حلقه.

Fig. 5. Block diagram of the HIL lab model.



شکل ۶- بلوک نمودار مدل صنعتی (عملیاتی) سخت‌افزار در حلقه.

Fig. 6. Block diagram of the industrial (operational) model of the HIL.

جدول ۱- تقسیم‌بندی زیرسیستم‌های شبیه‌سازی ماهواره‌بر براساس کاربرد.

Table 1. Classification of the satellite simulation subsystems based on application.

Subsystem	Embedded	RealTime	GUI	Windows
OFP	√	√	--	-
VLEPC	-----	-------	√	√
Monitoring	-----	√	√	√
VDS	-----	√	√	√
VDP (Vehicle Dynamic Plot)	-----	√	√	√

برنامه‌نویسی نرم‌افزار شبیه‌سازی

تلاش‌های اولیه در بررسی شبیه‌سازی، در تعریف سیستمی که باید مدل‌سازی شود و توصیف آن به وسیله‌ی نمودارهای گردش منطقی و روابط تابعی است. اما بالاخره با مسئله توصیف مدل به زبانی قابل قبول برای رایانه مواجه می‌شویم. تاکنون زبان‌های برنامه‌نویسی همه‌منظوره و تک‌منظوره زیادی ابداع و توسعه یافته که تعیین بهترین یا حتی کمی بهتر آن‌ها برای استفاده در یک مورد خاص تقریباً غیر ممکن است. در نتیجه، شیوه معمول انتخاب یک زبان، بهترین آن‌ها نبوده بلکه آشنایی تحلیلگر با آن است. هر نوع زبان الگوریتمی عمومی می‌تواند مدل دلخواه را بیان کند، اما ممکن است یکی از زبان‌های شبیه‌سازی تخصصی، دارای مزایای بسیار متمایزی از لحاظ آسانی، بازدهی وکارایی باشد [۱۷]. از دیدگاه سیستم عامل، آن‌ها را می‌توان به چهار گروه زبان‌های برنامه‌نویسی تحت Windows، تحت DOS، تحت سیستم‌های عامل منبع باز نظیر Linux و زبان اسمبلی (زبان ماشین) تقسیم کرد. بین سیستم‌های بلادرنگ سخت که نامطلوبیت یک الزام زمانی به شکست سیستم نهایی منجر می‌شود و سیستم‌های بلادرنگ نرم که گاه و بیگاه نامطلوبیت یک الزام زمانی قابل پذیرش است تفاوت وجود دارد.

در سیستم بلادرنگ نرم، درخواست‌ها به وسیله سیستم با بیشترین سرعت ممکن عملی می‌شود اما این درخواست‌ها در زمان مخصوصی به پایان نمی‌رسد. در سیستم بلادرنگ سخت، درخواست‌ها نه تنها به درستی انجام می‌شوند بلکه در زمان‌های خاصی به وقوع می‌پیوندند. اغلب سیستم‌های بلادرنگ به ترکیبی از هر دو نوع نیازمندند. اکثر سیستم‌های بلادرنگ، ادغام‌شده هستند. یک سیستم ادغام‌شده، رایانه‌ای است که درون سیستم ساخته شده و به‌وسیله کاربر به‌عنوان یک رایانه دیده نمی‌شود. یک سیستم ادغام‌شده هنگامی‌که شروع به کار می‌کند، بدون اینکه پایان یابد به اجرای Taskها می‌پردازد تا زمانی‌که خاموش شود. زیرسیستم‌های شبیه‌سازی 6DOF ماهواره‌بر براساس کاربرد، درجدول (۱) طبقه‌بندی شده است.

بنابراین با زیرسیستم‌هایی مواجه هستیم که هر کدام شرایط و معماری نرم‌افزاری خاصی را دارند. از این جدول واضح است معماری برنامه‌های زیرسیستم OFP به‌صورت ادغام‌شده طراحی می‌شود و چون سیستم‌های ادغام‌شده با کاربر در ارتباط نمی‌باشند. بنابراین این سیستم‌ها غالباً یا فاقد سیستم عامل هستند و یا دارای سیستم عامل DOS و یا یک سیستم عامل خاص نظیر RTLinux یا XPC Target می‌باشند.در یک سیستم ادغام‌شده استفاده از حافظه کمتر و سرعت محاسبات بالا و نیز برخورداری از قابلیت اطمینان بالا از اهمیت بسیار زیادی برخوردار است. بنابراین OFP را با هر زبان برنامه‌نویسی دلخواه نمی‌توان نوشت یا می‌توان نوشت ولی عملیاتی نخواهد بود.

یکی از نرم‌افزارهایی که برای نوشتن کد OFP مطرح است استفاده از محیط نرم‌افزاری Real Time Workshop شرکت Math Works است. لازمه استفاده از این نرم‌افزار این است که برنامه پرواز OFP بزبان Matlab یا Simulink نگاشته شود. سپس توسط یک مترجم C++ کامپایل گردد و برنامه کامپایل شده در یک سیستم ادغام‌شده که دارای سیستم عامل Target XPC است جای بگیرد. سیستم عامل Target XPC با زبان برنامه‌نویسی C++ نوشته شده است.

گرچه اکثر زبان‌ها و محیطشان ساختارهای یکدیگر را کپی می‌کنند ولی با نیازهای مختلف در ذهن تشکیل شده‌اند. هدف C++ توانمندی و کنترل در راستای پیچیدگی است. هدف دلفی برنامه‌نویسی ویژوال آسان بدون از دست دادن توانمندی وافر و ارتباط قوی با ویندوز است. هدف Java قابل حمل بودن است حتی با از دست دادن مقداری از سرعت و برنامه‌های توزیع شده یا محتویات قابل اجرای وب. زبان برنامه‌نویسی Ada رقیب بسیار قدرتمند و پرتوان‌تری از زبان C++ است. چرا که زبان Ada زبانی است که برای کاربردهای کلاس نظامی تهیه و تولید شده است بر خلاف C++ یک زبان ایمن است و پشتیبانی DOD را دارد اما چون در کشورمان امکان دسترسی به این زبان نرم‌افزاری وجود ندارد و در صورت دسترسی، ابزارهای این زبان با قیمت گزافی کرایه داده می‌شود، حتی در صورت تأمین منابع مالی، کاربران باتجربه در زمینه Ada نداریم، بنابراین بهتر است از آن صرف‌نظر شود.

زبان C و زبان++C با وجود قرابت‌های بسیار، اما دو زبان برنامه‌نویسی متفاوت هستند. زبان C برنامه‌نویسی ساخت‌یافته را پشتیبانی می‌کند درحالی‌که زبان C++ هم برنامه‌نویسی ساخت‌یافته و هم برنامه‌نویسی شی‌گرا را پشتیبانی می‌نماید. هر دو زبان، ارتباطشان با سخت‌افزار ساده است. هر چند تا این اواخر برای برنامه‌نویسی سیستم‌های ادغام شده به خصوص میکروکنترلرها از C استفاده می‌شد اما با تدوین استانداردهای لازم برای زبان C++ در آینده‌ای نزدیک اکثر چیپ‌ها و میکروکنترلرهای تجاری با زبان C++ برنامه‌ریزی خواهند

شد. بنابراین با توجه به قابلیت‌های فوق انتخاب زبان C++ در مقابل زبان C بدیهی است. سرانجام توصیه می‌شود OFP به زبان C++ نوشته شود چون C++ در مقایسه با زبان‌هایی مثل فرترن و پاسکال برای برنامه‌نویسی ادغام‌شده مناسب‌تر است. اما سایر زیرسیستم‌های شبیه‌سازی ماهواره‌بر را به‌خصوص VDS را می‌توان به زبان‌های فرترن، دلفی و غیره نوشت. اما در این صورت یکپارچگی کل برنامه شبیه‌سازی ماهواره‌بر از بین می‌رود ضمن آنکه دلایل دیگری نیز در انتخاب C++ برای VDS وجود دارد؛ از جمله آنکه فرترن و دلفی از نظر شی‌گرایی ضعیف هستند. چون اصولاً برای شی‌گرایی طراحی نشده‌اند. بنابراین، زبانی مانند فرترن علی‌رغم اینکه در گذشته، به عنوان زبان فعالیت‌های علمی و محاسباتی شناخته می‌شد، لکن در حال حاضر به دلیل عدم پشتیبانی مناسب و عدم سازگاری با مفاهیم (مانند شیءگرایی) و تکنولوژی‌های جدید (پشتیبانی از پردازنده‌های چند هسته‌ای و ۶۴ بیتی) یک زبان مرده محسوب می‌شود. همچنین در مورد دلفی موارد فوق صادق است.

برخی دیگر از زبان‌های جدید مثل JAVA برای نوشتن برنامه‌های بسیار بزرگ، Client/Server و تحت شبکه به کار می‌روند و برای فعالیت‌های علمی مناسب نیستند. JAVA خیلی شبیه C++ هست با این تفاوت که معایب C++ مثل پوینترها، ارث‌بری چندگانه، اپراتورها و غیره در آن حذف گردیده است. خیلی از بانک‌ها، خطوط هواپیمایی، مخابرات، وزارت نیرو و غیره از این زبان برای نوشتن برنامه‌هایشان استفاده می‌کنند. JAVA یک زبان قوی است و کاملا شی‌گرا، دارای قابلیت حمل بسیار بالا، امنیت عالی، مناسب برای پیاده‌سازی انواع معماری نرم‌افزار، کلاس‌های کتابخانه کامل و غیره است و نیز در حال حاضر توسط شرکت‌های بزرگ SUN و IBM پشتیبانی خوبی از آن و تکنولوژی‌های مرتبط با آن می‌شود. مترجم و ابزارهای توسعه (محیط کدنویسی و اشکال‌زدایی) متن‌آزاد و رایگان بسیاری برای آن وجود دارد. مهم‌تر از همه اینکه در حال حاضر تنها زبان مستقل از سکو است، به گونه‌ای که برنامه‌ای که کاملا با JAVA نوشته شده باشد بر روی هر سیستم‌عاملی قابل اجرا است. به‌رغم تمام این ویژگی‌های مثبت، به دلیل اینکه نهایتاً بر روی JVM اجرا می‌شود، کند است و برای کاربردهای شبیه‌سازی علمی مناسب نیست. (لازم به ذکر است برنامه‌های سازمانی مبتنی بر JAVA معمولا بر روی یک Application Container Server اجرا می‌شوند و بنابراین مشکل سرعت را ندارند). معایب اصلی JAVA سرعت پایین در اجرا و نیاز به Virtual Machine برای اجرا، قیمت زیاد برنامه‌ها و واسط کاربر تقریبا ضعیف است و با توجه به این معایب برای شبیه‌سازی 6DoF حامل ماهواره مناسب نیست.

اما از زبان C++ دو ورژن موجود است: Visual C++، که بر پایه زبان قدرتمند C (که برای کارهای سیستمی بسیار مناسب است) قرارگرفته و در نتیجه امکان استفاده از انبوه برنامه‌های مبتنی بر C را فراهم می‌کند، شیءگرا است و در حال‌حاضر پشتیبانی بسیار خوبی از جانب شرکت Microsoft از آن می‌شود و ابزارهای بسیار خوبی برای توسعه مبتنی بر آن وجود دارد (Microsoft Visual Studio 2008). این ابزارها از آخرین تکنولوژی‌های پردازنده‌های ۶۴ بیتی و چندهسته‌ای نیز پشتیبانی می‌کنند. C++ Builder، که همانند دلفی محیط کاربر پسند و راحت‌تری را نسبت به محیط Visual C++ در اختیار برنامه‌نویس قرار داده اما نسبت به Visual C++ از مترجم و Debugging ضعیف‌تری برخوردار است.

یادآور می‌شود، بیش از انتخاب زبان برنامه‌نویسی، تجربه برنامه‌نویسی بسیار مهم‌تر است. مرجع [۱۸] این مسئله را ثابت کرد که تجربه برنامه‌نویسی که برنامه‌ای را می‌نویسد از انتخاب زبانی که برنامه روی آن نوشته می‌شود در کارایی تاثیر بیشتری دارد و این بدان معناست که کارایی یک برنامه را برنامه‌نویس مشخص‌می‌کند و نه زبان برنامه‌نویسی.

استانداردهای Misra C و BARR-C برخی الزامات و توصیه‌هایی که برای تولید یک نرم‌افزار ادغام شده لازم است را ارائه کرده‌اند[۱۹]. با عرضه این استانداردها افق روشنی از به‌کارگیری زبان برنامه‌نویسی C++ جهت برنامه‌های OFP پیدا شد. در هر حال پیاده‌سازی استانداردهای Misra موجب می‌شود تست‌های استاتیکی کد برنامه و تعداد نقص‌های کد به حداقل برسد.

روند تکاملی شبیه‌سازی‌های نوین نشان می‌دهد که آن‌ها به سمت سامانه‌های پیچیده چند موضوعی پیش می‌روند؛ حوزه‌های الاستوپلاستیک، دینامیک برخورد ماورای صوت، کنترل فعال مواد هوشمند و ورودی‌های مبتنی بر عدم قطعیت، تقویت درک شهودی و کاربرپسندی از طریق رابط‌های کاربری هوشمند مؤثر از طریق واقعیت افزوده از چالش‌های شبیه‌سازی‌های آینده خواهند بود [۴].

نتیجه‌گیری و جمع‌بندی

در این مقاله استانداردهای لازم مهندسی نرم‌افزار برای تولید نرم‌افزار شبیه‌سازی ماهواره‌برهای چندمرحله‌ای صلب چند منظوره با رویکردی جدید معرفی گردید. ساختار پیشنهادی این قابلیت را به نرم‌افزار می‌دهد که به سرعت و با کمترین تغییرات در آزمایشگاه‌های نرم‌افزار در حلقه و سخت‌افزار در حلقه مورد استفاده قرار گیرد. رویه RUP برای تولید نرم‌افزار شبیه‌سازی 6DoF ماهواره‌بر پیشنهاد گردید. یادآوری شد که بیش از انتخاب زبان برنامه‌نویسی، تجربه برنامه‌نویسی بسیار مهم‌تر است. با توجه به تجربه خوب برنامه‌نویسی C++ در کشور، این زبان برنامه‌نویسی برای شبیه‌سازی 6DoF، سرعت نگارش و اشکال‌زدایی راحت‌تر پروژه بسیار کمک

خواهد کرد. پیشنهاد می‌شود در برنامه شبیه‌سازی 6DoF ماهواره‌بر تا آنجا که امکان دارد از C++ استاندارد استفاده شود، در این‌صورت برنامه خیلی به مترجم‌های C++ Builder، Turbo C++ و Visual C++ وابسته نخواهد بود و به‌راحتی با تغییرات جزئی در هر سه نوع مترجم قابلیت اجرا خواهد داشت.

تعارض منافع

هیچگونه تعارض منافع توسط نویسندگان بیان نشده است.

مراجع

[1] A. Tewari, *Atmospheric and Space Flight Dynamics: Modeling and Simulation with MATLAB and Simulink (Modeling and Simulation in Science, Engineering and Technology)*, Boston: Birkhäuser, 2007.

[2] G. Baldesi and M. Toso, "ESA launcher flight dynamics simulator used for system and subsystem level analyses," in *11th International Workshop on Simulation & EGSE Facilities for Space Programmes(SESP)*, 2010.

[3] G. Baldesi and M. Toso, "European space agency's launcher multibody dynamics simulator used for system and subsystem level analyses," *CEAS Space Journal*, vol. 3, no. 1, pp. 27-48, 2012, https://doi.org/10.1007/s12567-011-0023-9.

[4] M. Toso and V. Rossi, "ESA multibody tool for launchers and spacecrafts: Lesson learnt and future challenges," in *5th Joint International Conference on Multibody System Dynamics*, Lisbon, Portugal 2018.

[5] L. E. Briese, P. Acquatella B, and K. Schnepper, "Multidisciplinary modeling and simulation framework for launch vehicle system dynamics and control," *Acta Astronautica*, vol. 170, pp. 652-664, 2020, https://doi.org/10.1016/j.actaastro.2019.08.022.

[6] "A unified object-oriented language for systems modeling, language specification version 3.5." Modelica Assocition, Swedwn, 2021. [Online]. Available: https://modelica.org/documents/MLS.pdf

[7] I. M. Rodiana, U. Latifa, B. R. Trilaksono, E. Hidayat, and M. F. Sagala, "Software and hardware in the loop simulation of navigation system design based on state observer using Kalman filter for autonomous underwater glider," in *7th International Conference on Underwater System Technology: Theory and Applications (USYS)*, Kuala Lumpur, Malaysia, 2017, pp. 1-5, https://doi.org/10.1109/USYS.2017.8309461.

[8] I. Todić and V. Kuzmanović, "Hardware in the loop simulation for homing missiles," *Materials Today: Proceedings*, vol. 12, pp. 514-520, 2019, https://doi.org/10.1016/j.matpr.2019.03.157.

[9] A. A. Elgohary, A. M. Ashry, A. M. Kaoud, M. M. Gomaa, M. H. Darwish, and H. E. Taha, "Hardware-in-the-loop simulation of UAV altitude hold autopilot," in *AIAA SciTech Forum*, San Diego, CA, 2022, https://doi.org/10.2514/6.2022-1520.

[10] G. Marks, R. O'connor, M. Yilmaz, and P. M. Clarke, "An ISO/IEC 12207 perspective on software development process adaptation," *Software Quality Professional*, vol. 20, no. 2, pp. 48-58, 2018.

[11] *(ISO/IEC 12207) Standard for Information Technology - Software Life Cycle Processes*, ICS Code: 35.080 - Software, IEEE/EIA Standard, March 1998, https://doi.org/10.1109/IEEESTD.1998.88083.

[12] M. Jones, U. K. Mortensen, and J. Fairclough, "The ESA software engineering standards: past, present and future," in *International Symposium on Software Engineering Standards*, Walnut Creek, CA, USA, 1997, pp. 119-126. https://doi.org/10.1109/SESS.1997.595952.

[13] M. Dorfman and C. Anderson, "*Aerospace Software Engineering: A Collection of Concepts,"* American Institute of Aeronautics and astronautics, Inc., 1991.

[14] T. K. Tia, "Simulation model for rational unified process (RUP) software development life cycle," *SISTEMASI: Jurnal Sistem Informasi*, vol. 8, no. 1, pp. 176-184, 2019.

[15] J. S. Dahmann, F. Kuhl, and R. Weatherly, "Standards for simulation: As simple as possible but not simpler the high level architecture for simulation," *Simulation*, vol. 71, no. 6, pp. 378-387, 1998, https://doi.org/10.1177/003754979807100603.

[16] S. G. Tzafestas, *Applied Control: Current Trends and Modern Methodologies*. CRC Press, 1993.

[17] R. Shannon and J. D. Johannes, "Systems simulation: The art and science," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-6, no. 10, pp. 723-724, 1976, https://doi.org/10.1109/TSMC.1976.4309432.

[18] L. Prechelt, "Technical opinion: comparing Java vs. C/C++ efficiency differences to interpersonal differences," *Communications of the ACM*, vol. 42, no. 10, pp. 109-112, 1999, https://doi.org/10.1145/317665.317683.

[19] R. Bagnara, M. Barr, and P. M. Hill, "BARR-C: 2018 and MISRA C: 2012: synergy between the two most widely used C coding standards," *arXiv: 2003.06893*, 2020, https://doi.org/10.48550/arXiv.2003.06893.