ISSN 2354-0575

PHÁT TRIỂN MÔI TRƯỜNG MÔ PHỎNG CHO MÁY BAY TRỰC THĂNG KHÔNG NGƯỜI LÁI BẰNG C# VÀ X-PLANE

Lê Trí Quang*, Đỗ Thị Thu Trang
Trường Đại học Sư phạm Kỹ thuật Hưng Yên
** Tác giả liên hệ: quangletri@gmail.com*

Ngày tòa soạn nhận được bài báo: 15/09/2020
Ngày phản biện đánh giá và sửa chữa: 11/11/2020
Ngày bài báo được duyệt đăng: 15/01/2021

Tóm tắt:

Phương tiện không người lái nói riêng và thiết bị bay không người lái nói chung là tương lai của vận tải tự động. Do đó, chủ đề này thu hút được sự quan tâm của rất nhiều nhà nghiên cứu trên thế giới. Mục đích của bài báo này là kết hợp C# và X-Plane để phát triển môi trường mô phỏng cho máy bay trực thăng không người lái. Kết quả đạt được của đề tài có ý nghĩa quan trọng trong việc phát triển và hiện thực hóa bộ điều khiển cho máy bay trực thăng không người lái cỡ nhỏ phục vụ sản xuất và dân sinh, đồng thời, có thể sử dụng trong đào tạo sinh viên ngành Kĩ thuật Hàng không, ngành Kĩ thuật điều khiển và Tự động hóa.

Từ khóa: *Giải thuật điều khiển PID, môi trường mô phỏng SIL, máy bay trực thăng không người lái.*

1. Giới thiệu

Tiến hành thí nghiệm với máy bay trực thăng không người lái đòi hỏi rất nhiều thời gian, công sức, tiền của, đồng thời tiềm ẩn nguy cơ mất an toàn. Do đó, phát triển môi trường mô phỏng để đánh giá giải thuật điều khiển bay là cần thiết giúp rút ngắn thời gian, giảm chi phí trong quá trình nghiên cứu. Phổ biến có hai môi trường mô phỏng là Hardware-In-The_Loop (HIL) và Softwae-In-The-Loop (SIL). Trong bài báo này, nhóm tác giả trình bày giải pháp để phát triển môi trường mô phỏng SIL cho máy bay trực thăng không người lái. Thay vì sử dụng MATLAB&Simulink như đã đề xuất ở [1], toàn bộ chương trình được phát triển bằng C#. Động lực tiến hành nghiên cứu này, đó là, Tác giả mong muốn tạo ra một môi trường mô phỏng thuận tiện, giúp các nhà nghiên cứu có một môi trường mô phỏng tin cậy để tiến hành đánh giá thuật toán điều khiển sử dụng cho mục đích vận tải hàng hóa, cứu hộ, phun thuốc trừ sâu trong nông nghiệp…cũng như ứng dụng trong đào tạo các kĩ sư ngành hàng không và điều khiển tự động trong các trường đại học.

Với mục đích là phát triển môi trường mô phỏng bằng cách kết hợp C# và X-Plane, nên bài báo không đi sâu vào giới thiệu mô hình động học của máy bay trực thăng không người lái. Trọng tâm là trình bày các bước phát triển môi trường mô phỏng bằng C#, và hiện thực hóa giải thuật điều khiển PID để điều khiển cho máy bay trực thăng.

Phần còn lại của bài viết này được tổ chức như sau. Phần 2 trình bày các bước thiết lập và chương trình để phát triển môi trường mô phỏng. Phần 3 giới thiệu về bộ điều khiển PID, và lập trình bộ điều khiển PID thời gian rời rạc trên C#. Phần 4 là kết quả. Cuối cùng là kết luận và hướng phát triển.

2. Các nghiên cứu liên quan

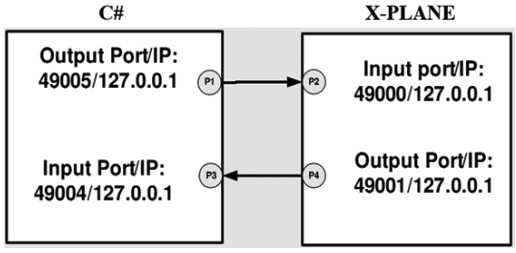
Trước đây các kĩ sư, các nhà nghiên cứu cần phải chế tạo nguyên mẫu một chiếc máy bay để nghiên cứu ảnh hưởng của luật điều khiển hoặc giải thuật dẫn đường mà họ phát triển. Các nguyên mẫu này được sử dụng đi sử dụng lại nhiều lần dẫn đến hư hỏng và phải chế tạo một nguyên mẫu khác. Mặt khác khi phải thay số bất kì một tham số nào đó, thì cần phải chế tạo lại một nguyên mẫu khác. Điều này gây tốn kém về thời gian và công sức. Để giải quyết vấn đề này, môi trường mô phỏng Hardware-In-The-Loop (HIL) và Software-In-The-Loop (SIL) được sử dụng để đánh giá giải thuật và điều khiển. Ưu điểm của giải pháp này là rút ngắn thời gian phát triển luật điều khiển, an toàn, tiết kiệm, linh hoạt dễ dàng thay đổi. Để xây dựng môi trường mô phỏng, nhiều giải pháp đã được đề xuất. Tuy nhiên, đa phần sử dụng MATLAB&Simulink để xây dựng môi trường mô

phỏng [1,2,3,4]. Do MATLAB&Simulink đã hỗ trợ các hàm kết nối, điều khiển và khả năng biểu diễn dữ liệu trên đồ thị nên giảm thời gian phát triển môi trường. Tuy nhiên, cần phải có bản quyền và tốc độ xử lí chậm cần phải trang bị máy tính có cấu hình cao. Vì vậy một số tác giả đã sử dụng ngôn ngữ lập trình C thay cho MATLAB [5], nhược điểm của giải pháp này là không có thư viện hỗ trợ sẵn, tuy nhiên tốc độ xử lý nhanh, dễ dàng mở rộng bằng cách kết nối với nhiều hệ thống nhúng khác nhau để đánh giá giải thuật và luật điều khiển.

3. Phát triển môi trường mô phỏng

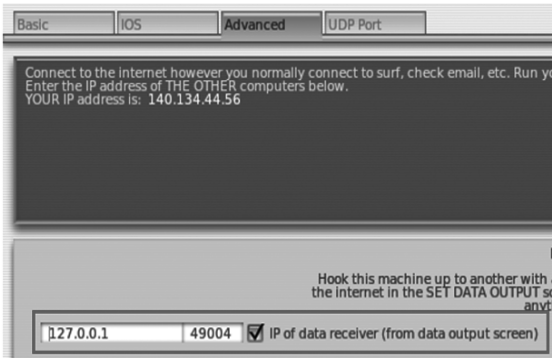
X-PLANE được cấp chứng nhận bởi trung tâm hàng không vũ trụ Hoa kì trong việc đào tạo phi công. Do đó, X-Plane cung cấp một môi trường tin cậy để các nhà nghiên cứu phát triển và kiểm tra tính khả thi của giải thuật điều khiển bay. Giải thuật điều khiển đã được đánh giá trên môi trường mô phỏng của X-Plane chắc chắn áp dụng được trên máy bay thực. Bởi X-Plane cho phéo thiết lập môi trường bay giống như thực tế bằng cách thêm các nhiễu loạn như gió, hoặc gió quẩn… Các lệnh điều khiển góc bay trên X-Plane được tuyến tính và chuẩn hóa như sau. Lệnh điều khiển góc tấn (pitch angle) và góc nghiêng (roll ange) có giá trị từ -1 đến 1. Lệnh điều khiển độ cao của máy bay có giá trị từ -12 đến 12. Lệnh điều khiển góc hướng (heading) có giá trị từ -16 đến 16.

3.1. Thiết lập cổng kết nối trên X-Plane

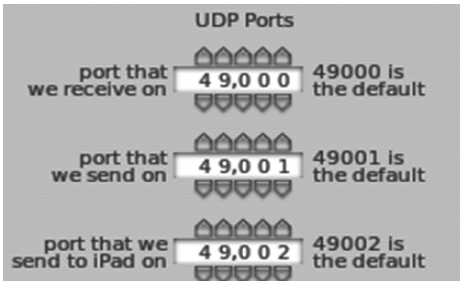


Hình 1. Kết nối giữa Simulink và X-Plane

X-Plane có thể trao đổi dữ liệu với môi trường bên ngoài thông qua gia thức UDP (User Datagram Protocol) [6]. Kết nối giữa C# và X-Plane được thực hiện thông qua mạng nội bộ có địa chỉ IP "127.0.0.1", như được thể thiện trong Hình 1. Để tạo kết nối này, cần phải thiết lập địa chỉ IP cho X-Plane như trong Hình 2, 3 dưới đây.



Hình 2. Thiết lập địa chỉ IP trên X-Plane



Hình 3. Thiết lập cổng kết nối trên X-Plane

3.2. Khung dữ liệu

Khung dữ liệu theo UDP bao gồm 41 byte, có cấu trúc như sau:

Byte	0	1	2	3
Giá trị	"D"	"A"	"T"	"A"

Trong đó:

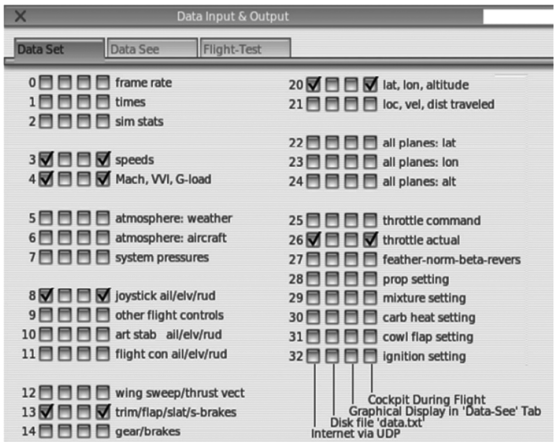
- Byte 0-3 là chuỗi kí tự "DATA" để thông báo bắt đầu một khung dữ liệu mới
- Byte 4 không được sử dụng cho truyền dữ liệu, thông thường giá trị của byte này có giá trị là 0
- 36 byte tiếp theo là dữ liệu gửi đi, trong đó 4 byte đầu là chỉ số của dữ liệu, và 32 byte cuối cùng là dữ liệu được biểu diễn dưới dạng số thực

Bảng 1. Cấu trúc khung dữ liệu UDP

DATA	I	Label	Group 1	Group 2	…
0-3	4	5-8	9-12	13-16	…

Để thu thập dữ liệu bay mong muốn phải chọn các tham số như Hình 4, lưu ý, số thứ tự 0, 1, 2… được gọi là nhãn (label).

Ví dụ, để gửi lệnh điều chỉnh chân ga (throttle) từ C# tới X-Plane, cần chọn label 25; để nhận giá trị của các tham số góc tấn (pitch angle), góc nghiêng (roll angle), hay góc hướng cần chọn label 18.



Hình 4. Chọn các tham số cần thu thập dữ liệu

2.2. Chương trình gửi/nhận dữ liệu trên C#

Để trao đổi dữ liệu giữa C# và X-Plane, có thể tóm tắt một số đặc trưng của UDP như sau: Tất cả dữ liệu được gửi theo tầng byte, mỗi khung dữ dữ liệu (data frame) có 41 byte, trong đó:

- 4 byte đầu tiên là thống báo "DATA"

- Byte thứ 5 là byte sử dụng nội bộ (internal-use) có giá trị là 0

- 36 byte tiếp theo là thông báo (message), trong đó:

+ 4 byte đầu tiên biểu thị nhãn (label) của dữ liệu

+ 32 byte tiếp theo là dữ liệu có định dạng là số thực có độ chính xác là 8 chữ số

Ví dụ một khung dữ liệu được gửi đi từ X-Plane có cấu trúc như sau:

68,65,84,65,60, 18,0,0,0, 171,103,81 191, 187,243,46,190, 103,246,45,67, 156,246,26,67, 47,231,26,67, 0,192,121,196, 0,192,121,196, 85,254,151,193

Ý nghĩa như sau:

- 68,65,84,65,60 = "DATA"
- 18,0,0,0 = 18 (label 18: pitch, roll, headings)
- 171,103 …, 193: là dữ liệu, trong C# có thể sử dụng hàm BitConverter.ToSingle để làm việc với kiểu dữ liệu này

Chú ý: Giá trị -999 hoặc 0,192,121,196 là trả lại quyền điều khiển cho X-Plane

Ví dụ, chương trình đọc dữ liệu từ X-Plane từ cổng 49004 (như thiết lập ở trên)

```
using System;
using System.Net;
using System.Net.Sockets;
namespace UDP_Receive
{
  class Program
  {
    static void Main(string[] args)
    {
      byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.
Any, 49004);
      UdpClient newsock = new UdpClient(ipep);
      Console.WriteLine("Waiting for a client...");
      IPEndPoint sender = new IPEndPoint(IPAddress.
Any, 0);
      Console.WriteLine("X-Plane Data Read: \n\n");
      data = newsock.Receive(ref sender);
      for (int index = 0; index < data.Length; index++)
      {
        Console.Write("{0},", data[index]);
      }

      Console.ReadKey(true); // Wait for any keypress
before closing
      newsock.Close();
    }
  }
}
```

Ví dụ, chương trình gửi dữ liệu tới X-Plane: chương trình này mục đích là để kiểm tra gửi lệnh điều khiển cho góc Pitch từ C# tới X-Plane. Khung dữ liệu gửi đi gồm 41 byte được khai báo trong mảng XFData, với 02 trường hợp: giá trị gửi đi có giá trị 0.02 (205, 204, 76, 62) và giá trị gửi đi có giá trị 0.0 (0, 0, 0, 0). Do chương trình chỉ muốn gửi lệnh điều khiển góc Pitch tới X-Plane nên các giá trị khác được thiết lập là 0, 192, 121, 196 (tương đương -999)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
namespace UDP_Send
{
  class Program
  {
    static void Main(string[] args)
    {
      byte[] data = new byte[1024]; // Array to hold
entire data string to be sent to X-Flight
      UdpClient server = new UdpClient("127.0.0.1",
49000);

      // --- 11: Flight Controls ---
      // Pitch: 0.20
      //byte[] XFData = { 68, 65, 84, 65, 0, 11, 0, 0, 0,
205, 204, 76, 62, 0, 192, 121, 196, 0, 192, 121, 196, 0,
```

```
192, 121, 196, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
        // Pitch: 0.00
        byte[] XFData = { 68, 65, 84, 65, 0, 11, 0, 0, 0,
0, 0, 0, 0, 0, 192, 121, 196, 0, 192, 121, 196, 0, 192, 121,
196, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
        float value;

       // Send the data to X-Plane
       server.Send(XFData, XFData.Length);
       // Print the data we sent to screen
       Console.WriteLine("Length: {0}", XFData.Length);
       Console.WriteLine("Data Set: {0}", XFData[5]);
       value = BitConverter.ToSingle(XFData, 9);
       Console.WriteLine("XFData1: {0}", value);
       value = BitConverter.ToSingle(XFData, 13);
       Console.WriteLine("XFData2: {0}", value);
       value = BitConverter.ToSingle(XFData, 17);
       Console.WriteLine("XFData3: {0}", value);
       value = BitConverter.ToSingle(XFData, 21);
       Console.WriteLine("XFData4: {0}", value);
       value = BitConverter.ToSingle(XFData, 25);
       Console.WriteLine("XFData5: {0}", value);
       value = BitConverter.ToSingle(XFData, 29);
       Console.WriteLine("XFData6: {0}", value);
       value = BitConverter.ToSingle(XFData, 33);
       Console.WriteLine("XFData7: {0}", value);
       value = BitConverter.ToSingle(XFData, 37);
       Console.WriteLine("XFData8: {0}", value);
       //-------------
       server.Close();
       Console.ReadKey(true); // Wait for keypress to close
program
     } // End Main

  } // End Program
}
```

2.3. Giao diện người dùng

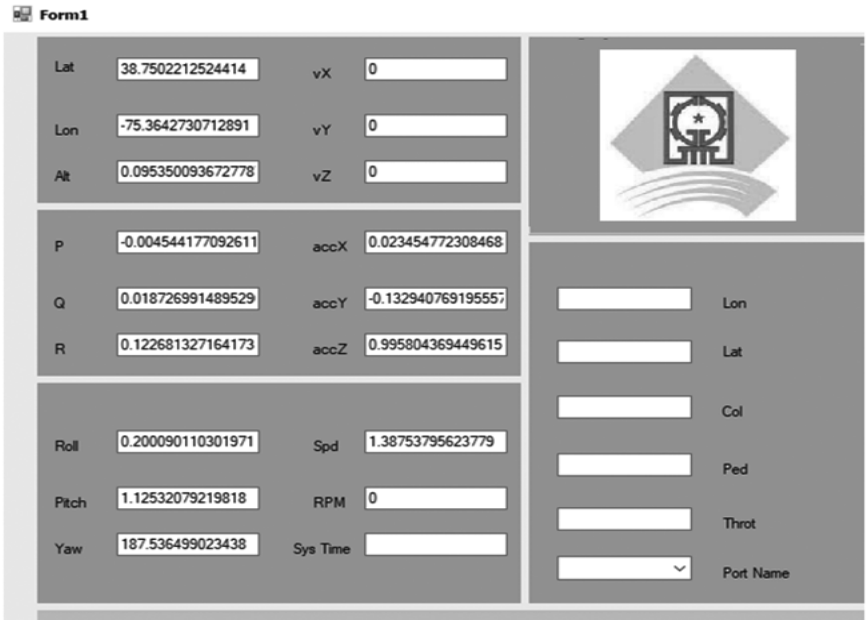
Để có thể quan sát các tham số, giao diện người dùng cần được phát triển, giả sử các tham số cần quan sát là vị trí của máy bay (Latitudinal (Lat), longitudinal (lon), altitude (alt)); gia tốc góc (p, q, r), vận tốc dài (vX, vY, vZ); gia tốc dài (accX, accY, accZ); góc lái (roll, pitch, yaw); các lệnh điều khiển (lat, lon, col, ped, throt). Như minh họa trong Hình 5, trao đổi dữ liệu giữa C# và X-Plane là hai chiều (đọc và gửi dữ liệu). Để thuận tiện cho việc đọc dữ liệu, ta nên sử dụng cấu trúc Struct để quản lí dữ liệu đọc về; lệnh gửi từ C# đến X-Plane được định dạng theo chuẩn của UDP. Các chương trình như sau:

Khai báo cấu trúc quản lí các biến đọc dữ liệu từ X-Plane

```
UdpDataStruct DataStruct = new UdpDataStruct();
   public struct UdpDataStruct
   {
     public double airspeed;
     public double Mach, accX, accY, accZ;
     public double P, Q, R;
     public double roll, pitch, yaw, heading_mag, mag_var;
     public double latitude, longitude, altitude;
     public double vX, vY, vZ;
     public double rpm;
     public double speedN, speedE;
   };
```

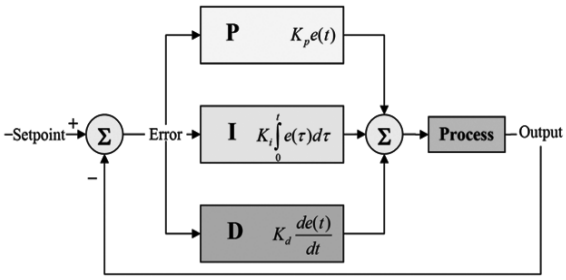
Khai báo cấu trúc quản lí các biến lệnh gửi tới X-Plane

```
public struct UdpCmdData
   {
     public float cmdPitch, cmdRoll, cmdYaw, cmdoffset,
cmdColl;
     public float desirePitch, desireRoll;
   };
```



Hình 5. Giao diện người dùng

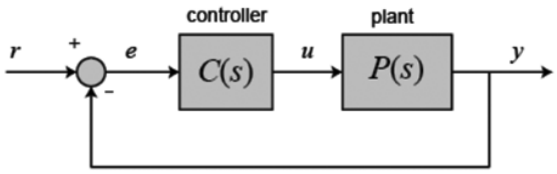
3. Bộ điều khiển PID



Hình 6. Cấu trúc bộ điều khiển PID

Bộ điều khiển tỉ lệ-tích phân-vi phân (proportional–integral–derivative controller (PID)) thuộc nhóm phương pháp điều khiển tuyến tính. Phương pháp điều khiển này được sử dụng rộng rãi trong công nghiệp. Ưu điểm của bộ điều khiển này là không yêu cầu mô hình toán học của hệ thống. Cấu trúc của bộ điều khiển PID được mô tả cụ thể trong Hình 7. Thành phần tỉ lệ P phụ thuộc vào sai số hiện tại; Thành phần tích phân phụ thuộc vào tích lũy các sai số quá khứ; Thành phần vi phân phụ thuộc vào tốc độ biến đổi sai số. Tổng của 3 thành phần này chính là giá trị đầu ra của bộ điều khiển. Ba tham số Kp, Ki, Kd sẽ được điều chỉnh để phù hợp với mỗi hệ thống cụ thể.

Hệ thống phản hồi đơn vị có cấu trúc như sau:



Hình 7. Cấu trúc hệ thống phản hồi đơn vị

Trong đó, r là tín hiệu đặt; e là sai lệch giữa giá trị đặt và giá trị thực tế; C(s) là bộ điều khiển; P(s) là hàm truyền; y là đầu ra của cơ cấu chấp hành. Nếu C(s) là bộ điều khiển PID trong miền thời gian liên tục thì tín hiệu điều khiển được tính như sau:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \quad (1)$$

Thực hiện biến đổi Laplace (1) trở thành

$$C(s) = K_p + \frac{K_i}{s} + K_d s \quad (2)$$

Trong thực tế, thành phần vi phân được biến đổi có dạng là bộ lọc thông thấp để giảm thiểu ảnh hưởng của nhiễu, do đó C(s) có dạng như sau

$$C(s) = K_p + \frac{K_i}{s} + \frac{NK_d}{1 + \frac{N}{s}} \quad (3)$$

Tuy nhiên, để có thể áp dụng bộ điều khiển PID ở biểu thức (3) trên các hệ thống nhúng (embedded system), cần phải biến đổi sang miền thời gian rời rạc. Giả sử, thời gian trích mẫu là Ts, qua phép biến đổi Backward Euler thành phần của tích phân và vi phân có dạng như sau:

Thành phần tích phân:

$$\frac{K_i T_s z}{z-1} \quad (4)$$

Thành phần vi phân

$$\frac{N(z-1)}{(1+NT_s)z-1} \quad (5)$$

Khi đó bộ điều khiển PID trong miền thời gian rời rạc có dạng sau:

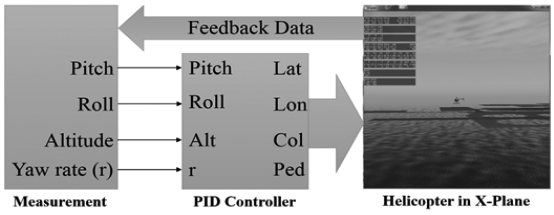
$$C(z) = K_p + \frac{K_i T_s z}{z-1} + \frac{N(z-1)}{(1+NT_s)z-1} \quad (6)$$

Chương trình PID trên C#

```
public double PID(double SP)
 {
   Pa0 = (1 + N * Ts);
   Pa1 = -(2 + N * Ts);
   Pa2 = 1;
   Pb0 = PKp * (1 + N * Ts) + PKi * Ts * (1 + N * Ts)
+ PKd * N;
   Pb1 = -(PKp * (2 + N * Ts) + PKi * Ts + 2 * PKd *
N);
   Pb2 = PKp + PKd * N;
   Pku1 = Pa1/Pa0; Pku2 = Pa2/Pa0; Pke0 = Pb0/Pa0;
Pke1 = Pb1/Pa0; Pke2 = Pb2/Pa0;
   //-----------------------------------------------------
   Pe2 = Pe1; Pe1 = Pe0; Pu2 = Pu1; Pu1 = Pu0; //
update variables
   pPV = RecData[1];
   Pe0 = SP - pPV;  // compute new error
   Pu0 = -Pku1 * Pu1 - Pku2 * Pu2 + Pke0 * Pe0 +
Pke1 * Pe1 + Pke2 * Pe2; // eq (12)
   if (Pu0 > PUMAX) Pu0 = PUMAX;  // limit to DAC
or PWM range
   else
      if (Pu0 < PUMIN) Pu0 = PUMIN;
   return (Pu0);   // sent to output
 }
```

4. Kết quả thử nghiệm mô hình

Hình 8 mô tả cấu trúc của hệ thống mô phỏng được phát triển, hệ thống này bao gồm ba khối cơ bản: khối đó lường (measurement), khối điều khiển PID, và máy bay trực thăng trên môi trường X-Plane. Trong đó, khối đo lường và khối điều khiển được phát triển trên môi trường C#. Trao đổi dữ liệu giữa C# và X-Plane thông qua giao thức UDP như đã trình bày ở trên. Hình 9 là kết quả khi điều khiển bay bằng bộ điều khiển PID từ C#.



Hình 8. Cấu trúc của môi trường mô phỏng SIL

Hình 9. Kết quả điều khiển bay cho máy bay trực thăng

5. Kết luận

Bài báo đã trình bày chi tiết giải pháp phát triển môi trường mô phỏng SIL cho máy bay trực thăng không người lái bằng cách kế hợp C# và X-Plane. Kết quả đạt được có ý nghĩa quan trọng trong việc phát triển và kiểm tra giải thuật điều khiển bay, cũng như ứng dụng vào giảng dạy cho sinh viên ngành Kĩ thuật Hàng không. Kết quả của nghiên cứu này cũng có thể sử dụng để nghiên cứu cho các loại máy bay khác như Drone, hay Fixed-Wing... Hơn thế nữa, do được phát triển bằng C# nên chương trình hoàn toàn có thể mở rộng để kế nối với các hệ thống nhúng bên ngoài giúp kiểm tra các phương pháp điều khiển trên các phần cứng khác nhau.

Lời cảm ơn

Nghiên cứu này được tài trợ bởi Trường Đại học Sư phạm kỹ thuật Hưng Yên trong đề tài mã số UTEHY.L.2020.26.

Tài liệu tham khảo

[1]. Bittar, Adriano, et al. "Guidance software-in-the-loop simulation using x-plane and simulink for uavs." *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2014.

[2]. Ribeiro, Lucio R., and Neusa Maria F. Oliveira. "UAV autopilot controllers test platform using Matlab/Simulink and X-Plane." *2010 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2010.

[3]. Jamadagni, Chinmayi S., et al. "System simulation approach for helicopter autopilot." 2014 *International Conference on Contemporary Computing and Informatics (IC3I)*. IEEE, 2014.

[4]. Figueiredo, Helosman V., and Osamu Saotome. "Simulation platform for quadricopter: Using matlab/simulink and x-plane." *2012 Brazilian Robotics Symposium and Latin American Robotics Symposium.* IEEE, 2012.

[5]. Bittar, Adriano, and Neusa Maria Franco De Oliveira. "Hardware-in-the-loop simulation of an attitude control with switching actuators for SUAV." 2*013 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2013.

[6]. https://en.wikipedia.org/wiki/User_Datagram_Protocol

DEVELOPMENT OF THE SIMULATION ENVIRONMENT FOR AN UNMANNED HELICOPTER BY USING C# AND X-PLANE

Abstract:

Unmanned aerial vehicles are the future of automated transport. Therefore, this topic has gotten the attention of many researchers in the world. This paper aims to develop the simulation environment for unmanned helicopters by using the C# and X-Plane. The obtained results are beneficial implications in the development and implementation of the controller for small scaling unmanned helicopter for production and daily life, as well as. It is useful in training for a student of Aviation Engineering, Control Engineering, and Automation.

Keywords: *PID, SIL, Unmanned helicopter.*