# DRIVING A CAD MODEL BY HARDWARE IN THE LOOP

**Konstantin Metodiev**

*Space Research and Technology Institute – Bulgarian Academy of Sciences*
*e-mail: komet@space.bas.bg*

***Keywords:*** *Autopilot Ardupilot Mega v.2, Autodesk Inventor, RC Engine MK-16, MPXV7002DP Static Pressure Transducer, Hardware in the Loop*

***Abstract:*** *In the paper hereby, a test study has been carried out as to whether semi-realistic experiments involving a 3D CAD model and a real sensor are feasible or not. For this purpose, a reputable CAD development environment is used to prepare the model. On the other side of the experimental setup, a static pressure transducer is mounted. The interfacing device is an autopilot widely used by hobbyists.*

*The targeted 3D model is a small-scale internal combustion engine intended for remotely controlled airplanes. The autopilot adjusts the blade pitch depending upon shaft revolutions and indicated airspeed (IAS). The former quantity is set in advance whilst the latter one (IAS) is derived by reading the static pressure transducer output through a 10-bit analog to digital converter.*

*The work sequence described this way lets developer adjust and validate a real control system parameters in terms of the 3D CAD model simulated response.*

# ПРОГРАМНО-АПАРАТНО УПРАВЛЕНИЕ НА CAD МОДЕЛ

**Константин Методиев**

*Институт за космически изследвания и технологии – Българска академия на науките*
*e-mail: komet@space.bas.bg*

***Ключови думи:*** *Автопилот Ardupilot Mega v.2, Autodesk Inventor, двигател MK-16, преобразувател на статично налягане MPXV7002DP, програмно-апаратно (полунатурно) моделиране*

***Резюме:*** *В настоящия доклад е описан експеримент, за да се установи дали полунатурното моделиране с използване на 3D CAD модел и реален сензор е правдоподобно или не. За тази цел е използвана известна развойна CAD среда, с помощта на която е създаден модела. От другата страна на експерименталната установка е монтиран преобразувател на статично налягане. Свързващото устройство е автопилот, масово използван в авиомоделизма.*

*Целевият 3D модел е двигател с вътрешно горене, предназначен за безпилотни летателни апарати. Автопилотът настройва стъпката на витлото в зависимост от оборотите на вала и индикаторната въздушна скорост. Оборотите са предварително зададени, докато въздушната скорост се получава от преобразувателя на статично налягане и 10-битов аналого-цифров преобразувател.*

*Работната последователност, описана по този начин, позволява на разработчика да настройва и валидира параметрите на реална система за управление въз основа на реакцията на 3D CAD модел в симулатора.*

## Introduction

Hardware-in-the-loop simulation is an alternative way of testing and validating complex real-time embedded systems which are extensively used in automotive systems, robotics, radar, power systems, etc., [1], [2]. Among all advantages that simulation provides, following might be outlined: reduced cost, enhanced safety, repeatability, meeting tight deadlines during design stage, and others.

Broadly speaking, it is only appropriate to run a simulation if carrying out an equivalent real experiment is either too expensive or impossible. What motivates making the presented study, however, is implementing a genuine algorithm of a semi-realistic simulation by means of Autodesk®

Inventor. The widely renowned CAD software is an integrated development environment which provides support to engineers and technicians during the process of machine design. Notwithstanding that a variety of standards, design modules, and simulation tools are available within it, Autodesk® Inventor lacks an electric simulation capability, probably on purpose. A hardware in the loop simulation might presumably evade this shortage which is yet another purpose of the presented study case.

The paper hereby gives a detailed explanation of how to perform a test of both an embedded system (sensor and autopilot) and a plant (CAD model) in real time. It does not serve as a conclusive evidence of utilizing the mentioned equipment. Instead, an exemplary way of working out a solution to the assigned task is thoroughly described including schematics, references to movies, and source codes. An alternative description done with care and much detail, in author's humble opinion, would be difficult to encounter elsewhere.

## Materials and methods

*Hardware in the loop*

The experimental setup is outlined in Fig. 1. It consists of NXP Semiconductors MPXV7002DP static pressure transducer @ ±2 kPa full scale and autopilot Ardupilot Mega v.2.0. The analog to digital converter turns sensor analog output into quantization levels. The autopilot sends result further to a PC through the serial UART to USB converter.

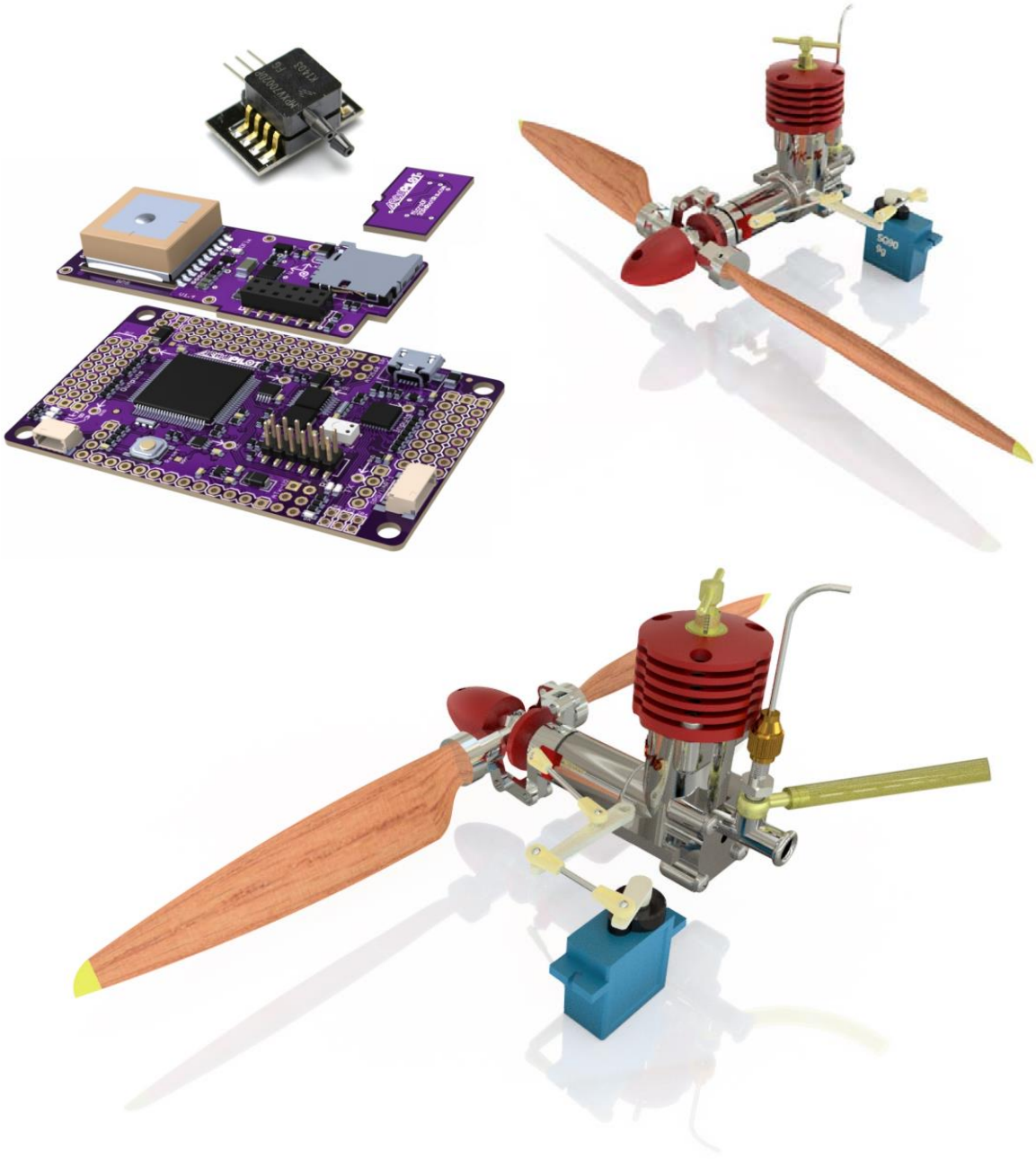

Fig. 1. Hardware used in the loop and targeted 3D model

*CAD model*

In addition, Autodesk® Inventor has been used to prepare a 3D model and develop an iLogic rule. The model is Russian MK-16 internal combustion engine which is shown in Fig. 1. The engine is upgraded with a variable blade pitch device which is solely designed by the author. The other engine parts are authentic. A second-class lever is mounted between the servo motor and the pitch altering mechanism in order to provide the servo with mechanical advantage and ensure mechanism stability and irreversibility at high pitch values as well. The iLogic rule reads the serial port within a timeout interval and adjusts the blade pitch according to shaft angular speed and indicated airspeed.

*Code snippets*

In order to put the CAD model in motion, two source code snippets had to be developed. These are shown in the Appendix section. The source code intended for the autopilot is developed by means of Arduino IDE. The driving iLogic rule is developed by means of iLogic editor within Inventor IDE employing Visual Basic for Applications syntax. Both the PC and the autopilot exchange information through the serial port.

The main functions of the iLogic rule are to scan the serial port within a timeout interval, update the mechanism status, and refresh the display. Upon successful opening the serial port, it is advisable to encompass the main program flow by a finite loop in order to let the program reach the "Close-Serial-Port" method at end of the snippet. The rule reads the serial port within a try/catch block in order to conceivably process a timeout exception. Both source codes (Ardupilot and iLogic) are self-explanatory and might be found in the Appendix.

*Calibration*

The MPXV7002DP static pressure transducer outputs a single voltage depending upon the pressure difference at sensor's ports. The sensor output signal relative to pressure input might be found in product datasheet [3] and Fig 3. The working media is dry air only.

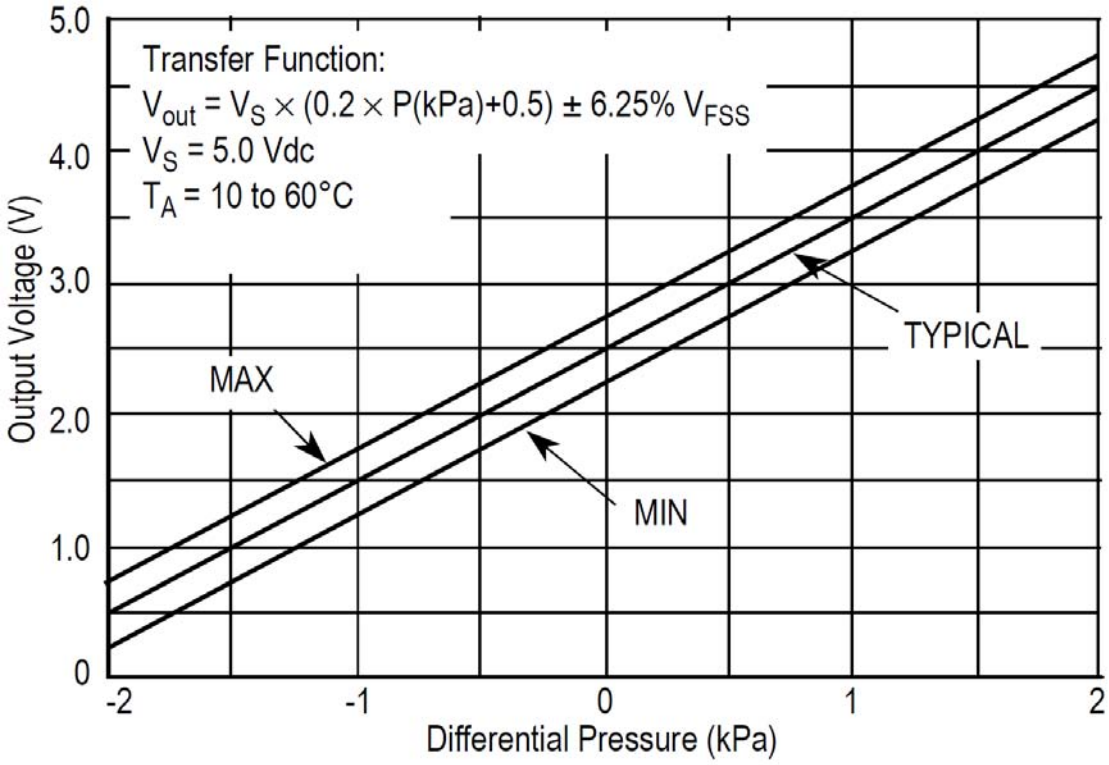


Fig. 2. Transducer MPXV7002 response [3]

*Blade pitch*

An automatic pitch device adjust the blade so as to align the airfoil chord with hypotenuse of the so-called velocity right triangle, Fig. 3. The triangle catheti are formed by forward velocity and peripheral velocity vectors. The forward velocity is equal to the measured airspeed whilst the peripheral velocity depends on blade section radii and shaft angular velocity. The assembly consists of linkages that alter the blade pitch. Simultaneously, the entire assembly rotates alongside the engine

shaft. The only exception is the stator which connects the mechanism to the actuating servo motor. The mechanism is antisymmetric in relation to the engine shaft, Fig. 3.

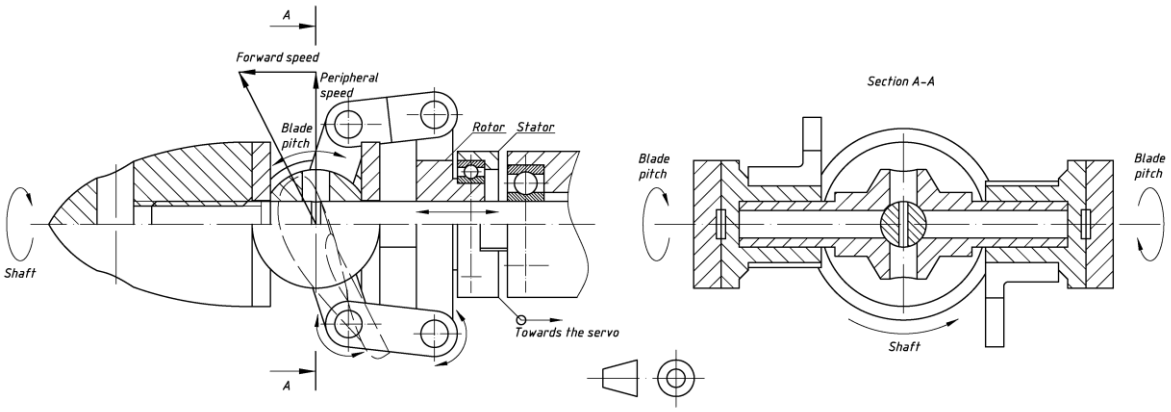


Fig. 3. Basic understanding of the proposed blade pitch altering device

*Putting it all together*

The indicated airspeed might be derived in terms of a difference between total (stagnation) pressure and static pressure values. In this case, both quantities are taken by a Pitot tube which in turn delivers these pressures to the sensor input orifices. The static pressure transducer provides a single voltage output depending on the pressure difference. In order to compute the airspeed, following formula is used for incompressible fluids:

$$u = \sqrt{\frac{2(p_t - p_s)}{\rho}} \quad (1)$$

where $p_t$ is the stagnation pressure, $p_s$ is the static pressure, kg/(m.s$^2$); ρ is the fluid density, kg/m$^3$; and u is the fluid velocity, m/s.

The autopilot sends integers from 0 to 1023 through the serial port. The iLogic rule processes these values in order to restore the airspeed. Having had the full scale output and zero offset, the developer may convert the quantization levels back to voltage. Finally, the calibration curve given in [3] might be used in order to restore the pressure difference $p_t - p_s$. The fluid density adopted in this case is 1.2 kg/m$^3$.

**Results**

A brief demonstration of the project might be watched by following both links [4] and [5] in the References section. A baud rate of 9600 bps is quite enough to make the CAD model react to sensor stimuli in a timely manner. The experiment itself is shown in Fig. 4.

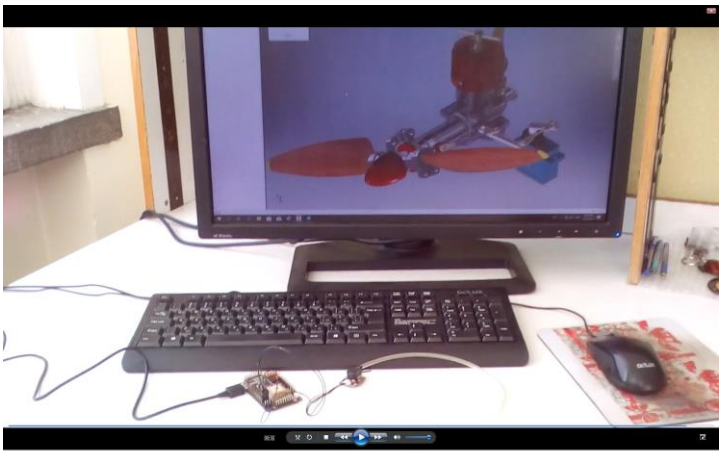

Fig. 4. A picture taken during semi-realistic simulation

As it was already mention before, the pitch altering leverage mechanism is antisymmetric in relation to engine crankshaft. Therefore, the mechanism is not balanced statically. This in turn may introduce additional vibrations at high angular speeds. Same is true for the propeller itself. In order to verify the aforementioned assumption, a dynamic simulation was carried out in addition. A torque of 0.5 N.mm was applied to the engine shaft and the gravity vector was set. A prominent point was selected to measure the ongoing accelerations with. The results are shown in Fig. 5. The graph shows total linear acceleration values that the designated point undergoes during the test run. No significant vibrations could be discerned.

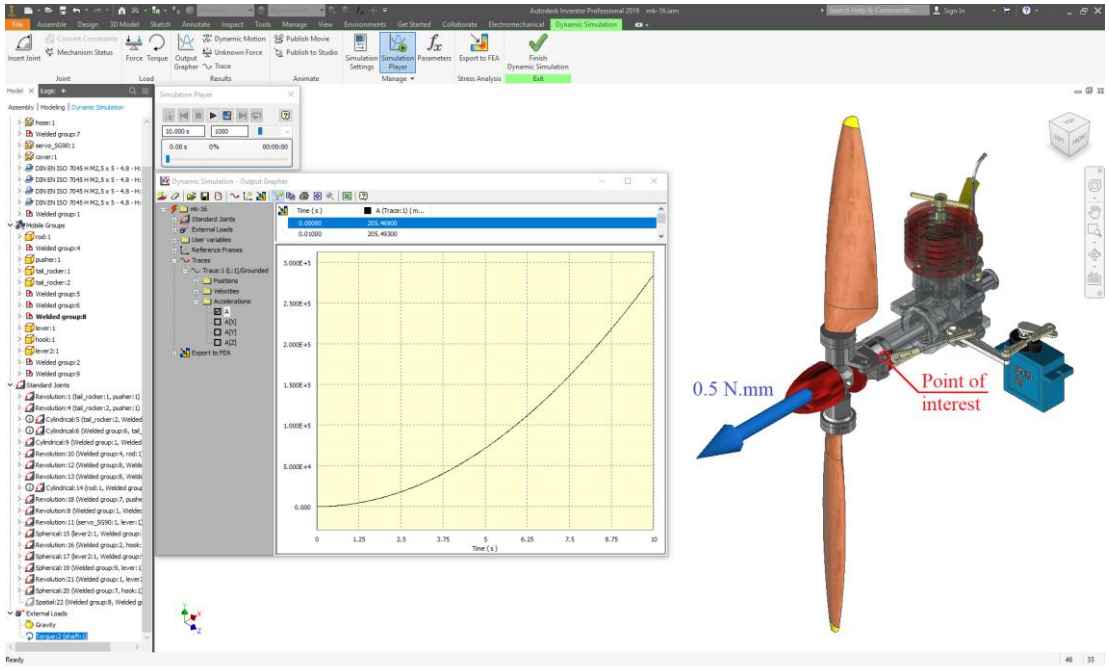


Fig. 5. The engine assembly put to the test in the Dynamic Simulation module

**Discussion**

The presented project is merely demonstrative. It aims at showing yet another feature of Inventor that hasn't been used on a large scale yet. Contemporary machines are equipped with numerous electronic parts. Unfortunately, Autodesk Inventor does not provide an ability to simulate electrical equipment operation of any kind. The presented project suggests a workaround. In addition to the proposed blade pitch altering device, the semi-realistic simulation could be run alongside a large variety of designs, such as robotic arms, radar, power, offshore systems, [1]. In the presented study, the developer is able to attune the controlling embedded system with reference to the 3D CAD model response.

The presented iLogic rule might be developed further by triggering an interrupt service routine whenever serial data are available. The current solution scans the port on a regular basis employing the so-called "polling" technique which is less effective, yet it is easier for programming contrariwise.

It is also worth mentioning that using sensors with digital output ($I^2C$, SPI, etc.) might facilitate carrying out the experiment to a large extent. In this case, the skill demanding calibration stage is redundant. The sensor parameters such as precision, accuracy, zero shift, etc. depend mostly on the manufacturer. The user only needs to read and interpret available data.

Arduino offers a free version of a C/C++ compiler, so does Autodesk with regard to Inventor, i.e. a 30-days fully functional trial copy which was used in the presented study.

The MK-16 engine model is available for download at the author's page in GrabCAD [6].

**References:**

1. https://en.wikipedia.org/wiki/Hardware-in-the-loop_simulation
2. Kleijn, C., Introduction to Hardware-in-the-Loop Simulation, Controllab Products B.V., the Netherlands.
3. MPXV7002 Integrated Silicon Pressure Sensor On-Chip Signal Conditioned, Temperature Compensated, and Calibrated, NXP Semiconductors, Data Sheet, Document Number: MPXV7002, Rev 4, 03/2017, p. 6.
4. https://www.youtube.com/watch?v=8-HqrnOJYwY
5. https://www.youtube.com/watch?v=RffqRiiP2Mo
6. https://grabcad.com/library/an-internal-combustion-engine-1

**Appendix**

*A.1. Source code for Ardupilot Mega v.2.0*

```
int sensorPin = A0;
int sensorValue, offset = 0;

void setup() {

int i;

for (i = 0; i < 10; i++) {
sensorValue = analogRead(sensorPin) -
              512;
offset += sensorValue;
} // for_i
offset /= 10;

Serial.begin(9600, SERIAL_8N1);

} // setup

float Vout = 0.;
void loop() {

sensorValue = analogRead(sensorPin) -
              offset;

Serial.println(String(sensorValue,
DEC));
delay(125);

} // loop
```

*A.2. Source code for iLogic rule*

```
Sub Main()
Dim myComPort As IO.Ports.SerialPort = Nothing
Dim temp As Double = 0

dist = 1
rot = 0

myComPort =
My.Computer.Ports.OpenSerialPort("COM4", _
9600, 0, 8, 1)
myComPort.ReadTimeout = 250
MessageBox.Show("Press OK to start logging", _
"ReadCOM")

For i = 1 To 300
rot -= 10
temp = CDblAny(GetSerialData(myComPort))
If IsNumeric(temp) Then dist = Round(temp / _
1024 * 5, 1)
RefreshScreen()
Next i

dist = 1
rot = 0

myComPort.Close()
End Sub

Sub Delay_ms(msec)
System.Threading.Thread.CurrentThread.Sleep(msec)
End Sub

Sub RefreshScreen()
RuleParametersOutput()
InventorVb.DocumentUpdate()
ThisApplication.ActiveView.Update()
End Sub

Function GetSerialData(ByRef theComPort As _
IO.Ports.SerialPort) As Double
Dim returnStr As String = ""
Dim returnVal As Double = 0
Try
returnStr = theComPort.ReadLine()
If returnStr Is Nothing Then
MessageBox.Show("Empty string", "Read Com")
Else
Double.TryParse(returnStr, returnVal)
End If
Catch ex As TimeoutException
returnStr = "Error: Serial Port read timed out"
Finally
End Try
Return ReturnVal
End Function
```