Airship as an Interesting Tool for the Solving of Various ICT Tasks - Design and Implementation of Testing Interface

Matej Jakab, Adrián Kováčik, Michal Hodoň, Peter Ševčík

***Abstract*—** This paper deals with the usage of an airship as a tool for solving various tasks of information and communication technologies (ICT). The most common tasks are measurement of air pollution parameters, monitoring and video recording. There is a big potential in energy savings against quadcopters. An airship can do its job multiple longer times compared to the very common quadcopters. This document describes design and implementation of testing interface. Interface is used for testing controlling model of airship from viewpoint of user. Model of airship in Blender application is used for emulating movement of real airship.

***Keywords*—** airship, OpenRex, stabilization, accelerometer, magnetometer, tilt compensation, electronic compass, Blender, visualization..

I. Introduction

Nowadays, the utilization of quad- , hexa-, or octacopters (etc.) for different monitor-ing tasks is quite common to many people. However, it is almost impossible to ob-serve an airship. Though a couple of interesting application scenarios was proposed and successfully applied till this time, as mentioned in [1] - [12], due to the implemen-tation feasibility, utilization of robotic airships is still rare. It should be however said, that airships deliver a big potential in energy savings against, e.g. quadcopters. An airship can do its job multiple longer times compared to the very common multicopters. Depending on the used airship size, it can carry sen-sors to measure air pollution parameters or a camera for video recording. The use of an airship equipped with various tools is wide. It can measure the atmosphere, ob-serve traffic or wildlife. One of the commercial uses is to live stream a video of for an example a concert or any other cultural event. Whenever an accident happens, the airship can observe the place surroundings. Stabilization of an airship requires obser-vations of its position. When the airship is set to a position to stay at, and the position has been changed, the stabilization module must detect these changes and react. In this part, we have tested FXOS8700CQ sensor, a magnetometer combined with an accelerometer. We were sending the resulting data to the Blender software to visualize the results.

II. Stabilization

We must think of different cases of problems related to the airship stabilization. Different methods are used for this purpose on the background of tuned sensory subsystem, [13] - [15].

To move the airship, we must know the heading, its pitch and its roll. We must know this, because when we want to move the airship, we must turn the airship using the steering propeller on the back and we must turn the propelling motors to adjust the pitch, too. Then we can move the airship up or down, back or forward. To know the heading, the pitch, the roll we will follow this document. First we will meat the theory, then the practical part.

During an observation mission, the airship must stay on a desired position. If a wind blows or the airship does not stay on the place for any other reason, it must return to that position. To find out the current position we will use a GPS module connected to the main board. Another

Matej Jakab, University of Žilina, Slovak Republic (e-mail: jakab@stud.uniza.sk)
Adrián Kováčik University of Žilina, Slovak Republic (e-mail: kovacik@stud.uniza.sk)
Michal Hodoň, University of Žilina, Slovak Republic (e-mail: michal.hodon@fri.uniza.sk)
Peter Ševčík, University of Žilina, Slovak Republic (e-mail: peter.sevcik@fri.uniza.sk)

case, when the airship will have to get back to a position it was is, when it lost the LTE signal. In that case we must have previous position saved.

Another feature implemented is similar to car parking sensors, but more ultrasonic distance sensors must be used, because we are covering a 3D space instead of the 2D parking system. The driver has two options to choose. First, the airship will stop when the sensors detect a critical distance between the airship and another object. The second option is that the driver is allowed to hit the object. At slower speed, a collision with some other objects is not always damaging. It is because the airship is light, and it floats.

III. Electronic compass

We have decided to use the OpenRex primary platform as the main control unit of airship. The reason is that OpenRex has many sensors including a gyroscope, an ac-celerometer and a magnetometer. We can connect a disc via the SATA interface. To use LTE, here is a PCIE mini slot and a micro SIM slot, on the board. The OpenRex board is running a Linux with a custom file system. FXOS8700CQ Sensor is a magnetometer combined with an accelerometer. The sensor was used together with GPS for the navigation purposes. To keep an eye on the values of the pitch, the roll and the electronic compass head-ing, we visualize the data in the Blender software. We get the data via the USB port, which is connected with the OpenRex board. In the Blender software we run a Python script to read the data and update the rotations of the X, Y, Z axes of a block, which is representing the board. The serial communication must be set correctly on sides, the python script and the C script running on the OpenRex board.

A compass is a device which points to the north. An electronic compass is similar device based on a magnetometer sensor. Using the electronic compass we can figure out which direction and how much is the airship turned from the north. To know in which direction the airship is heading, we have used the sensor FXOS8700CQ sol-dered down on the OpenRex board. We can use this as an electronic compass. Be-cause the accelerometer is here, too, we can tilt compensate the compass, otherwise we would be only able to use the magnetometer compass along a one single plane, and if tilt, the data will not be accurate.

When the airship tilts, the airship axes change. The same happens with the magne-tometer. Because of that, we must tilt compensate the electronic compass. To do so, we apply the formulas using normalized values of the accelerometer. Using these values it is possible to calculate the pitch and the roll of the airship. According to the pitch and the roll values, it is possible to calculate tilt compensated values. However, this will work approximately up to +40 degrees or –40 degrees of the tilt.

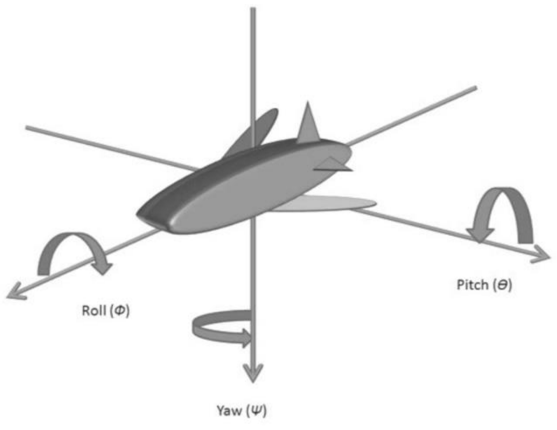


Fig. 1. The Airship Tilt Axes

The source of hard iron distortion is materials, which produce a constant magnetic field. It is additive to the Earth`s magnetic field. For example a permanent magnet or a mag-netic deposit close to the sensor, can produce a hard iron distortion. Soft iron distortion is a distortion coming from materials, which do not generate a direct magnetic field. The materials can have an influence on the Earth`s magnetic field and that way they distorts the magnetic field. It is for example nickel or iron. This distortion is not always a big problem and sometimes it is not needed to filter the soft iron distortion. In the case we are going to filter this distortion, we apply it after the tilt compensation and after the hard iron distortion. The soft iron distortion is more complicated compared to the hard iron, therefore it can´t be removed with a single constant but we need a more complicated filter formula. Magnetic declination is the difference of the magnetic and the astronomic north of the Earth. Here the value is approximately +4,5 degrees, in Žilina. This value must be added at the end to have the correct compass result.

IV. Control algorithm

To read, convert the sensor data into usable units and send it, we are running a C script using an I2C communication interface. We send the data to the PC via a serial communication interface. At the beginning of the program we set up the I2C, the serial port and the sensor registers. If it is needed we have to re-calibrate the magne-tometer. Then, in the infinite loop, we read the register data, correct the data and send them via the serial interface. Here we can see the raw magnetometer and accelerometer output:

```
Magnetometer raw data:
magRawX: 103, magRawY: 250, magRawZ: -293
```

```
Accelerometer raw data:
accRawX: -44, accRawY: 14, accRawZ: 4060
```

Now we are going to work with these raw values to get the heading of the board so we can use it as the compass. If we want an accurate electronic compass, we must apply the following steps:

Calibration - During the magnetometer calibration, to get its maximal and minimal values, we turn the board in all the axes. To skip the calibration next time we use this device, we have to save and use the values. This is guaranteed only if we use the magnetometer at the same place.

Tilt compensation - In the beginning we must calculate the normalized accelerometer data:

```
accNormX = accRawX / sqrt( accRawX2 + accRawY2 + accRawZ2 );
accNormY = accRawY / sqrt( accRawX2 + accRawY2 + accRawZ2 );
```

The normalized values of the accelerometer are close to 0. It is because the OpenRex board is not tilted but it is lying on a flat table:

```
Accelerometer normalized data:
accNormX: -00,01
accNormY: 000,00
```

We calculate the pitch:

```
pitch = asin( accNormX );
```

We calculate the roll:

```
roll = -asin( accNormY / cos(pitch) );
```

We can see the calculated pitch and the roll values lower here:

```
pitch: -00,01
roll: -00,00
```

Now we can calculate the compensated values of the tilted magnetometer:

```
magCompX = magRawX . cos(pitch) + magRawZ . sin(pitch);
magCompY = magRawX . sin(roll) . sin(pitch) + magRawY . cos(roll) - magRawZ
. sin(roll) . cos(pitch);
```

Hard Iron Distortion - To get rid of the hard iron distortion we must apply the following formula:

```
magDataX = magCompX - ( magMinX + magMaxX ) / 2;
magDataY = magCompY - ( magMinY + magMaxY ) / 2;
```

Soft Iron Distortion - Sometimes we must filter the soft iron distortion. To do so, we apply this formula:

```
magDataScaledX = ( magDataX – magMinX ) / ( magMaxX – magMinX ) . 2 – 1;
magDataScaledY = ( magDataY – magMinY ) / ( magMaxY – magMinY ) . 2 – 1;
```

One of the last steps is to calculate the magnetometer heading. Depending on the use of the soft iron filter, we use the variable magData or the magDataScaled:

```
heading = 180 . atan2( magDataY, magDataX ) / 3,14159265;
```

Magnetic Declination - Finally, to correct the heading we must add the magnetic declination to the heading:

```
heading = heading + declination;
```

Lower here we can see the result as the heading of the magnetometer:

Tilt compensated electronic compass with added declination:

```
heading: 066,14
```

Pitch and Roll in Degrees - We can calculate a little bit more. Out from the accelerometer raw data we can calcu-late the pitch and the roll in degrees:

```
angleX = ( atan2( accRawY, accRawZ ) + 3,14159265 ) . 57,29578;
angleY = ( atan2( accRawZ, accRawX ) + 3,14159265 ) . 57,29578
```

We can observe the X and Y angles calculated by the upper formulas:

```
Angles calculated out from the accelerometer raw data:
angleX: 180,20, angleY: 180,62
```

V. TESTING INTERFACE

We decided to use as testing interface, application Blender, which is open source software used for modeling and rendering 3D computer graphic. Another strong as-pect of this application is possibility of writing and executing scripts written in Python programming language and utilization of bpy.py library provided by Blender, which extends abilities of Python programming language by incorporating functions, which provide access to data, functions and tools of program Blender. Testing interface of Blender was used by us to create model of airship.

We didn’t come out from any particular specification during process of modeling and for purposes of testing was our airship model very simple. When specification of concrete construction and positions of engines will be known, it is possible to adjust model according specifics. Our designed model is sufficient for emulating purposes. The more important aspects of model as its appearance are model attributes in meaning of Blender application. Each model in Blender is perceived as object in this application with many different attributes, such as size, position, location etc. The most important attributes for us was position and location of object. Position of ob-ject is its relative position in consideration of center of coordinate system defines as point (0,0,0). Our model is depicted at beginning of emulation. Second important attribute of object is rotation of object, which describes position of object in consid-eration of each individual axis x, y, z.

Object interaction is made by user in user interface of Blender application or through execution of Python script, where object and its attributes are accessible with bpy library.

A. Design and implementation of Python script

Script design for controlling movement of virtual airship must fulfill several condi-tions, which we determined. Those conditions were:

- Simple controlling of model airship by utilization of functions, that provides bpy library
- Creation of connection and reading controlling data from primary applica-tion
- Autonomous execution of Python script from the rest of Blender applica-tion, because of fluid execution of this application

Design of script consisted of three main aspects, with which we provided all de-clared conditions. Our designed solution is only one from many, which can be done by Python scripting. This solution is sufficient for purposes of our project and its im-plementation permits later extensions of functionalities. Those three aspects of scripts were definition and implementation of modal operator class, implementation of separated thread and implementation of local TCP/IP server inside that thread.

B. Modal operator class

Class Operator (bpy.Types.Operator) defines class, in which it is possible to create and there are created tools of Blender application. Epithet modal has those tools, which implements function of same name and examples of those tools are transla-tion, rotation and other interactive tools, which are used for manipulation with object and its attributes in real time. Supported by those knowledge and abilities of basic class Operator, we designed inherited class

AirshipOperator, which task was auton-omous controlling of all manipulation with model based on data from external source. Independence and undisrupted execution of operator instructions is provided by keyword return values of each individual function of operator, where each function return value is decided according to inner logic. Relevant keyword return values for us during design were:

- Running modal keyword is used for defining call of method modal(), which is executed parallel with Blender application and its execution is repeated until method is cancelled or task is finished;
- Finished keyword is used to terminate execution of operator, when operator fulfilled its purpose;
- Cancelled keyword is used to terminate execution of operator, when error or exception occurred during execution of one of the function;
- Pass through keyword is passive return value, after which operator waits un-til one of Blender events cause call of modal() function.

Implementation of inherited class AirshipOperator had to contain following func-tions for providing expected functionality:

- Execute() or invoke() functions are used to initialize created object of opera-tor and return value in case correct execution of those function is Running modal;
- Modal() method, which was implemented with translation and rotation in-teraction with object of airship model (for purposes of fluid interaction with user interface of Blender application, we implemented all required reading of controlling data from primary application and processing of all required calculation inside separated thread)

C. *Method modal()*

Method modal() is called after occurrence of any type of event in application, concretely events, which occur in active window with object. Those event can be caused by inputs from keyboard, mouse or through activation of timer. In execute() method of operator, we registered during initialization of object timer with command, which invokes timer event each 10 milliseconds (timer periodicity can be simply changed according our predefined requirements), and method modal is executed.

```
self.timer = context.window_manager.event_timer_add
(0.01, context.window)
```

Method modal is executed also after invocation by any other type of event, which is cause by other component. It is impossible to influence appearance of other events, but we can filter out other event through condition inside modal method. In the next example, it can be seen snippet of modal method source code, where object attributes of location and rotation are set, in case event type is timer.

```
if event.type == 'TIMER':
bpy.data.objects['Airship'].location =
self.thread.position
bpy.data.objects['Airship'].rotation_euler =
self.thread.rotation
```

As it can be seen from code snippet, we used functionality of bpy library, which made objects and its attributes available. Bpy library consists of different modules. We used several of them in our script and those modules incorporate:

• bpy.context - contains information about actual working environment, such as actual working area, screen, scene, window and window manager ;
• bpy.data - provides access to Blender internal data and objects ;
• bpy.ops - provides python access to calling operators, this includes operator written in C, Python and also our own defined operators ;
• bpy.types
• bpy.utils – contains utility functions specific to Blender but not associated with blenders internal data .

With those abilities of bpy library, it is easily possible to manipulate many different things in Blender application, control and handle several different objects.

D. *Thread implementation*

We implemented separated thread class standardly, evenly as in Python language, inherited from Python threading.Thread class. Main purpose of implementing thread was to lighten main application from reading controlling data and their processing. Thread class was implemented with several methods with different purport.

1) __init__(self) method: contains initialization of thread object and setting of da-ta variables to initial values,

```
self.position = [0,0,0]
self.rotation = [0,1.57,0]
```

where variable position defines actual position of object in coordinate system and variable rotation is its orientation in consideration of axes. Inchoative value of posi-tion variable is center of coordinate system, where emulation starts. Unit, in which variable elements are indicated, is internal unit of measurement in Blender applica-tion. Initial value of rotation variable sets object to horizontal and lateral orientation in consideration of user viewpoint. To achieve that, we had to set initial offset of 1,57 (or π/2) radian to Y axis. Rotation variable elements are in radian unit.

2) start(self) method: contains assignment of positive value to running variable, which stores information about thread activity and it is accessible from main applica-tion thread. Second task of method is starting thread itself.

3) stop(self) method: contains assignment of negative value to running variable, which has impact to existence of thread. Method is called from main application thread, when emulation is over.

4) run(self) method: contains all thread logic and incorporates while loop, which maintains and executes thread commands. Essential tasks of method are:

a) Communication: Creation and maintaining of local TCP/IP socket server, on which primary application connects and sends controlling data

b) Data reading: Data are read in periodic intervals and temporary stored be-fore processing and transformation, in form of data packet, which contains con-trol data. We defined data packet in simplified way, which is sufficient for purpos-es of emulating movement. Implementation of more complicated moveset in-cludes redesign of data packet and processing of new data. Content of our data packet and meaning of its elements is:

Data packet			
Direction axis movement data	*Altitude* axis movement data	*Translation* movement data	*Translation* movement data

- First data packet element (Direction data) contains information about ob-ject movement in horizontal axis given the surface
- Second data packet element (Altitude data) contains information about ob-ject movement in vertical axis given the surface, ascending or descending of whole object or its part, which depends on behavior of real airship model
- Third and fourth packet elements (Translation data) contain instruction for initialization or continuation of object translation movement forward, where all attributes of position and rotation are used to determine movement in coordinate system.

Data elements in data packet are separated with colon, which is used to ascer-tain each element independently. Each data element value is from range <0,100>. Direction and altitude data elements passive value is 50, where increment or dec-rement of this value leads to rotation movement. Translation data elements pas-sive value is 0, where its increment lead to translation movement forward.

c) Data transformation: Processed data packet is used for transformation into object movement in Blender environment. First and second data elements con-tains exact requirement for rotation in one corresponding axis. Value of data ele-ment is used to determine direction and length of rotation. Rotation movement and altitude regulation of model object are done through modification of its rota-tion attribute in corresponding axis.

```
if int(dataElem[0]) <= 20:
self.rotation[2] += 0.01
if int(dataElem[0]) > 20 and int(dataElem[0]) <= 40:
self.rotation[2] += 0.005
```

Processed data from data packet are stored in dataElem array variable and ac-cording their value transformed to object rotation in one concrete axis. Change of rotation attribute is small, because of high frequency of repeating control data read-ing and their transformation. Translation movement forward is more complicated, due to fact that object movement occurs in tridimensional space, which means in each of its axes. Resulting translation line depends on initial position of object and its orientation in consideration of axes. We implemented calculation of such movement through goniometric functions,

```
self.position[0] -= 0.006 * math.cos(self.rotation[1] -
1.57) * math.cos(self.rotation[2])
self.position[1] -= 0.006 * math.cos(self.rotation[1] -
1.57) * math.sin(self.rotation[2])
self.position[2] += 0.006 * math.sin(self.rotation[1] -
1.57)
```

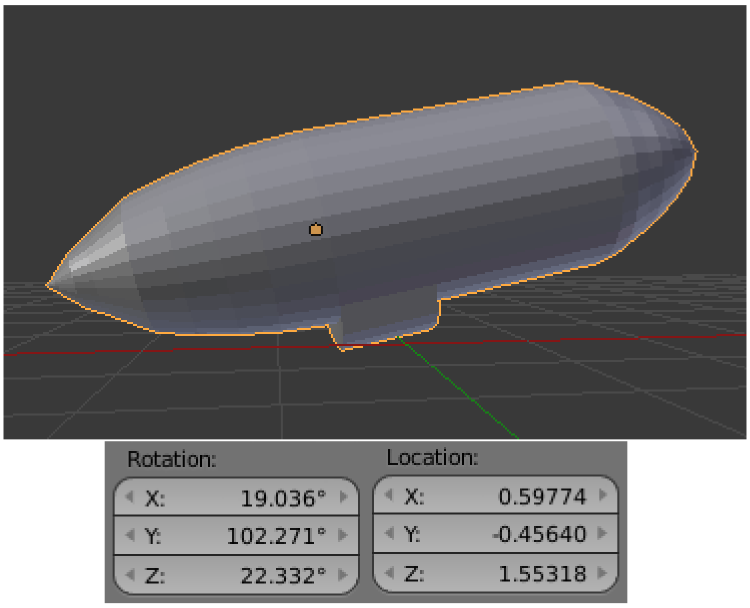
where position array defines actual position of object in all axes and new position is calculated through goniometry with consideration of actual orientation of object.

E. Local TCP/IP socket server implementation

One of the reasons, why we chose application Blender as core application for our testing interface and emulation of airship movement was possibility to utilize script-ing with Python language. Primary application used by user is defined in real situation as client and software on real airship as server, on which client connects. We could define communication interface with equivalent attributes in testing interface, through Python scripting and implementation of local TCP/IP socket server with equally to server interface on airship. This way, we comprehended in our design not only emulation of airship movement, but also simplified emulation of communica-tion interface. Server is defined as TCP/IP socket communication, through stream.

```
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.bind(('127.0.0.1', 9013))
```

Server listens on localhost adress (‘127.0.0.1’) and waits for connection from primary application on same computer, which features as client part of communi-cation interface.



Rotation:		Location:	
X:	19.036°	X:	0.59774
Y:	102.271°	Y:	-0.45640
Z:	22.332°	Z:	1.55318

Fig. 2. Airship model in Blender application and attributes of rotation and location

VI. Conclusion

Introduced testing interface was successfully implemented and verified. It will be used as the simulation tool with the possibilities of quick proof of different control policies applicable for the airship movement. Such tool will significantly save costs during the development process of the airship navigation algorithm. It also open an-other options for integration of different physical models which could improve the simulation scenario to be as close to the real conditions as possible. Portability of the system allows easy transmission of developed algorithms between virtual world and the airship with mentioned control unit.

Acknowledgment

"This publication was realized with support of Operational Program Integrated Infrastructure 2014 - 2020 of the project: Intelligent operating and processing systems for UAVs, code ITMS 313011V422, co-financed by the European Regional Development Fund".

EUROPEAN UNION
European Regional Development Fund
OP Integrated Infrastructure 2014 – 2020

MINISTRY
OF TRANSPORT
OF THE SLOVAK REPUBLIC

REFERENCES

[1] Y.-b. Li and X. Wang, "Ground station software design of tethered air-ship monitoring system," *2011 International Symposium on Computer Science and Society*, IEEE, Kota Kinabalu, Malaysia, Jul. 16-17, 2011, DOI: 10.1109/ISCCS.2011.99.

[2] A. Elfes, S. S. Bueno, M. Bergerman, and J. G. Ramos, "A semi-autonomous robotic airship for environmental monitoring missions," *IEEE International Conference on Robotics and Automation*, Leuven, Belgium, May 20, 1998, DOI: 10.1109/ROBOT.1998.680971.

[3] J. J. SantaPietro, "Persistent wide area surveillance from an airship," *IEEE Aerospace and Electronic Systems Magazine*, vol. 27, no. 6, pp. 10-16, Jun. 2012, DOI: 10.1109/MAES.2012.6328548.

[4] K. Izet-Ünsalan and D. Ünsalan, "A low cost alternative for satellites - tethered ultra-high altitude balloons," *5th IEEE International Conference on Recent Advances in Space Technologies*, Istanbul, Turkey, Jun. 9-11, 2011, DOI: 10.1109/RAST.2011.5966806.

[5] T. Xu, X. Zhang, and Y. Lu, "Onboard controlling system design of unmanned airship," *International Conference on Electronic and Mechanical Engineering and Information Technology IEEE*, Harbin, Heilongjiang, China, Aug. 12-14, 2011, DOI: 10.1109/EMEIT.2011.6023728.

[6] Y. Liu, Z. Pan, D. Stirling, and F. Naghdy, "Control of autonomous airship," *IEEE International Conference on Robotics and Biomimetics*, Guilin, China, Dec. 19-23, 2009, DOI: 10.1109/ROBIO.2009.5420403.

[7] M. Galletti, G. Krieger, T. Borner, N. Marquart, and J. Schultz-Stellenfleth, "Concept design of a near-space radar for tsunami detection," *IEEE International Geoscience and Remote Sensing Symposium*, Barcelona, Spain, Jul. 23-28, 2007, DOI: 10.1109/IGARSS.2007.4422723.

[8] H. A. Kadir and M. R. Arshad, "The development of unmanned small-size double hull blimp for low altitude surveillance system," *IEEE International Conference on Underwater System Technology: Theory and Applications*, Penang, Malaysia, Dec. 13-14, 2016, DOI: 10.1109/USYS.2016.7893922.

[9] D. Gobiha and N. K. Sinha, "Autonomous maneuvering of a stratospheric airship," *IEEE Indian Control Conference*, Kanpur, India, Jan. 4-6, 2018, DOI: 10.1109/INDIANCC.2018.8307998.

[10] P. Sun, X. Wang, and W. Xie, "Centrifugal blower of stratospheric airship," *IEEE Access*, vol. 6, pp. 11516-11524, Feb. 27, 2018, DOI: 10.1109/ACCESS.2018.2809707.

[11] G. Franceschetti, V. Gervasio, and F. Vinelli, "Large urban areas monitoring: Use of unmanned microwave powered airships as remote sensing nodes," *International Microwave Workshop Series on Innovative Wireless Power Transmission: Technologies, Systems, and Applications*, Kyoto, Japan, May 10-11, 2012, DOI: 10.1109/IMWS.2012.6215796.

[12] J. Zahour, L. Elis, J. Krivka, P. Stetka, and K. Kosturik, "Control unit of monitoring airship and its communication interface," *21st Telecommunications Forum*, Belgrade, Serbia, Nov. 26-28, 2013, DOI: 10.1109/TELFOR.2013.6716298.

[13] J. Wang and M. Xiuyun, "Active disturbance rejection backstepping control for trajectory tracking of the unmanned airship," *IEEE International Conference on Unmanned Systems*, Beijing, China, Oct. 27-29, 2017, DOI: 10.1109/ICUS.2017.8278379.

[14] B. Chen, "Adaptive backstepping sliding mode tracking control for the stratospheric airship," *12th International Conference on Computer Science and Education*, Houston, TX, USA, Oct. 30, 2017, DOI: 10.1109/ICCSE.2017.8085489.

[15] F. Ben Abdallah, N. Azouz, L. Beji, and A. Abichou, "Modeling and stabilization of a cable-driven parallel platform suspended by an airship," *11th International Workshop on Robot Motion and Control*, Wasowo, Poland, Jul. 3-5, 2017, DOI: 10.1109/RoMoCo.2017.8003892.

[16] P. Herman and W. Adamski, "Nonlinear tracking control for some marine vehicles and airships," *11th International Workshop on Robot Motion and Control*, Wasowo, Poland, Jul. 3-5, 2017, DOI: 10.1109/RoMoCo.2017.8003922.

[17] C. Blouin, E. Lanteigne, and W. Gueaieb, "Optimal control for the trajectory planning of micro airships," *International Conference on Unmanned Aircraft Systems*, Miami, FL, USA, Jun. 13-16, 2017, DOI: 10.1109/ICUAS.2017.7991324.

[18] C. Nie, M. Zhu, Z. Zheng, and Z. Wu, "Model-free control for stratospheric airship based on reinforcement learning," *35th Chinese Control Conference*, Chengdu, China, Jul. 27-29, 2016, DOI: 10.1109/ChiCC.2016.7555054.