Applied Computer Science, vol. 12, no. 2, pp. 5–21

Submitted: 2016-06-02
Revised: 2016-06-10
Accepted: 2016-06-18

mobile robots, python computer language, programming

Andrzej PIOTROWSKI[*]

AN ANALYSIS OF THE USE OF THE PYTHON LANGUAGE IN ROBOT APPLICATIONS

Abstract

The paper has analyzed the possibility of using the Python computer language in programming mobile robots. The designed and constructed test stand – a caterpillar mobile robot, enables the testing of wireless control algorithms, responses to events, I/O device programming methods and path selection algorithms. The software applications(client and server) written in the Python computer language have been developed and analyzed. A partial listing of the program source is included. The useful-ness of Python in robot applications was confirmed.

1. INTRODUCTION

The end of the 20th century and the beginning of the 21st century mark a period of the expansion of autonomous robots. The development of computer technologies has contributed to an increase in productivity with the simultaneous miniaturization of computerised systems. Control systems have become smaller and capable of being installed in small-sized devices. An example are drones (Fig. 1) – robotized flying machines of a size from several dozen centimetres to several meters [11, 17]. Particularly small versions are popular on the consumer market and are used for filming object or invigilating people. In the case of ope-rational drones, in addition to vision systems, the drones are equipped with advanced combat systems that enable the carrying out of "surgical" attacks on chosen targets. As a rule, drones are remotely controlled by the operator, who is dozens of kilometres away from the battlefield, using radio waves (most often via a satellite connection). This means that most important decisions are theoretically made by a human.

[*] Czestochowa University of Technology, Al. Armii Krajowej 21, 42-201 Czestochowa, apiotr@itm.pcz.pl

However, in the event of a disturbance or break in connection, the robot's control system and its operating software are able to independently make a decision on the continued operation of the drone (to retry to make connection with the operator, automatically return to the base, or to launch an attack on the selected target). Autonomous robots offer also a method for exploring the space. Small-sized and a few hundred grams in weight, mobile robots provide an excellent alternative to dangerous manned missions. Power supplied from photovoltaic batteries and furnished with advanced control systems and exploration tools, they are able to practically independently conduct geological surveys or to analyze the composition of the atmosphere on a remote planet. An example are the successful Mars exploration attempts with the use of the US Sojourner or Curiosity rovers (Fig. 1a) [11, 17].

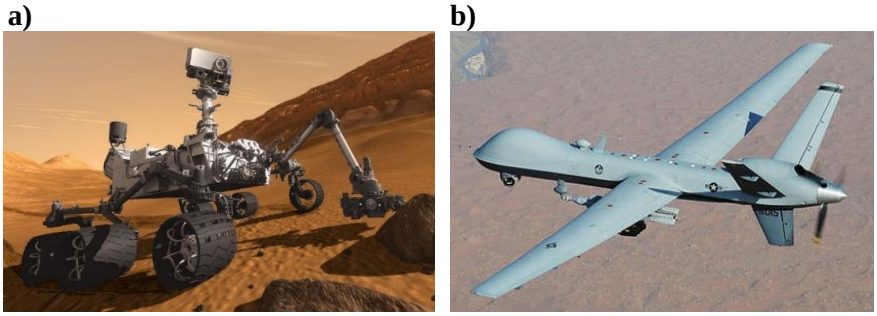

Fig. 1. a) Curiosity – The Mars Rover – A visual concept [17], b) MQ-9 Reaper [11]

Autonomous robots are constructed, as a rule, based on advanced electronic subassemblies. The first models used PLC controllers adapted to the purposes of robotics and were programmed based on languages typical for industrial applications, e.g. graphical languages, such as the Sequential Function Chart (SFC) language, the Function Block Diagram (FBD) language and the Ladder Diagram (LD) language; and text languages, such as the Structured Text (ST) language and the Instruction List (IL) language. The use of text languages allowed possible reprogramming of the robot that was situated at an immense distance from Earth, as it required short and easy-to-compress text messages. The next stage in the development of robotic systems were control systems relying on micro-controllers. For their programming, programming languages from the C and C++ families and, for simple applications, the adapted BASIC language were used. Created applications were then compiled and stored in the memory of the controller or microcontroller. Their characteristic feature was a small capacity of the operating memory, which required short applications to be created. The most recent solutions and the immense miniaturization of computer systems allow computers to be used in robot systems, which are furnished with advanced, highly efficient processors, a large operating memory and numerous I/O ports enabling the control

of a huge number of external devices. Unlike microcontrollers, a classic operating system known from home use is possible to be run on a such a computer. This is a great asset. The use of a traditional operating system from the Microsoft Windows or Unix\Linux family allows alternative programming languages to be used. For example, the very popular Raspberry Pi microcomputer uses the Linux operating system, while the pre-ferred language for the operation and programming of the GPIO connector is the Python language [5, 16]. This constitutes a total departure from the traditional perception of programming robots as a very complex task to be carried out by advanced programmers using the C language or its derivatives.

By observing the changes occurring in the construction and programming of robots, the possibilities of using the Python language in mobile robot applications [11, 14, 19] were examined. The investigation was divided into several stages. The first stage included the selection, construction and building of a test stand; the second stage covered the design and making of a control systems based on the state-of-the-art microcomputer systems; the third stage encompassed the writing of a client-server standard control application using the Python language; and the four stage summed up the experiment results. All the above-mentioned work was completed at the Institute for Mechanic Technologies of the Czestochowa University of Technology with a contribution by the members of scientific associations active at the Institute.

2. THE TEST STAND – A CATERPILLAR MOBILE ROBOT

2.1. Mechanical construction

In terms of their design solutions, autonomous mobile robots can be divided into two groups. The first group includes vehicles using a classic or modified wheel system with swivel wheels allowing the choice of the direction. The second group consists of caterpillar vehicles that use an independent drive of two mutually parallel caterpillars. The control of each caterpillar allows changing the vehicle's movement direction without having to construct a complex steering system, which is characteristic of wheeled vehicles. This simplifies the construction of the robot and facilitates its control. When building the test stand, a decision was made to construct a caterpillar vehicle driven by two independent stepper motors. The construction of a small robot of dimensions of 200 x 180 x 120 mm and a weight of about 4 kg was assumed. These dimensions were determined by the size of the control system which was built based on a Raspberry Pi microcomputer in version 2 [8, 13, 15, 20].

The robot was designed within the SolidWorks 2015 software program by 3DS. Several conceptual versions of the robot were developed in the form of three-dimensional CAD models. Using SolidWorks' advanced programming

functions, the analysis of the design and kinematics of the caterpillar vehicle was made, which enabled the selection of the simplest and cheapest solution. The design was simplified to a maximum. The originally assumed system of spring shock absorbers and tensioners was abandoned for simple rockers that performed two functions: shock absorbing and caterpillar tensioning. The drive system was simplified to use only five wheels: two large road wheels (front and rear), two small rocker-mounted wheels and a single driving wheel positioned above the rear wheel, connected with the propulsion motor with a rack system (Fig. 2a). It should be noted, at the same time, that for the construction of a heavier larger-size model, the original concept relying on damped road wheels (with torsion springs) and a classic front wheel and a rear driving wheel will be used.

a) **b)**

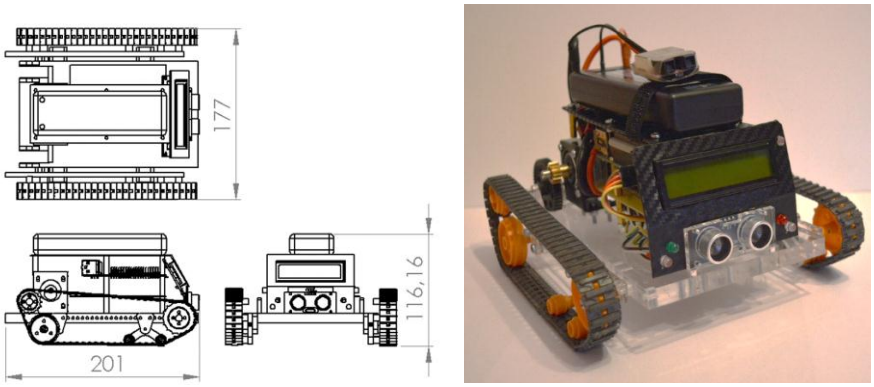

Fig. 2. The test stand – a caterpillar mobile robot [source: own study]

The basic constructional element is a floor panel with brackets supporting the drive system and control system elements. It was made from plastic (acrylic glass) using a laser cutting technology, which ensured a high accuracy and aesthetics of robot execution (Fig. 2b). Distance sensors, a camera and illumination elements to light up the space around the vehicle were mounted on the floor panel brackets [2,12]. The robot is additionally furnished with an LCD screen for testing purposes. It is used for displaying commands received by the rover's control system (the response to the received command). This enables a very fast analysis of any errors without having to logging on and verifying the file containing information on the operation of the software and the control system (error log).

2.2. The control system and electric & electronic elements

The heart of the control system employed in the test stand is a Raspberry PI 2 computer. This is the second version of the Raspberry Pi mini-computer developed in 2012 in the UK by the Raspberry Foundation [7]. The size of a credit card, built based on an ARM-architecture processor being typical rather of mobile phones, the microcomputer turned out to be a great success. In spite of its, at first, low computing performance, it became a standard in building of, not only amateur, control systems. Its greatest asset distinguishing it from other constructions of this type is a 40-pin GPIO connector (I2C, SPI, UART interfaces) terminated outside, which enables a huge number of peripheral devices (such as stepper motors, controllers, robot control systems, etc.) to be connected. The PI 2 Raspberry system used on the test stand is distinguished, compared to the first version, by a more efficient processor and the RAM memory expanded to 1 GB (table 1).

Tab. 1. The characteristics of the Raspberry PI 2 microcomputer [14]

Processor	Broadcom BCM2836 quad core Cortex A7 @ 900MHz
GPU	VideoCore IV
Memory	1GB SDRAM
Card reader	Memory card reader micro SD
Outputs	HDMI 1.4, AV (*3.5mm jack*)
LAN	Ethernet 10/100Mb
Ports	4x USB 2.0 ports, 1x micro USB port
Extensions	40 pins GPIO, camera and monitor connector
Power	5V from micro USB port

A feature that qualifies the Raspberry PI 2 microcomputer to be used in mobile systems is its low energy demand. The system is supplied with a voltage of 5V and a (maximum) current intensity of 2A from a LiPol Redox Racing 4000 mAh battery pack (Fig. 3), which is sufficient for 2 hours of operation at the maximum energy consumption (operating motors, the additional lighting switched on, continuous communication between the Client and the Server, 100% loading of the Cortex A7 processor). Batteries of this type are characterized by a large discharge current, regardless of the battery pack charging level. Therefore, the current efficiency does not decreases with the decrease in voltage, which results in stable power supplying [12].

All the system elements are connected to the Raspberry PI 2 GPIO socket (Fig. 3). These are two SY35ST28-0504A stepper motors controlled by A4988 systems. The motors drive the robot's caterpillars via a reduction gear (two gear wheels). For the assessment of the distance from an obstacle, an HC-SR04 ultrasonic sensor is used, which measures the distance in the range of 20 to 2000 mm

with an accuracy of 3 mm. It was assumed in the design that the robot would be controlled by the operator. However, for safety reasons (communication breaking, no visibility) and for testing purposes (testing the automatic path selection algorithms), the robot was equipped with a system for determining the distance from any objects being in its way. It has been provided for in the developed algorithm that in case of no reaction on the operator's part, the robot will automatically stop at a distance of 30 mm before the obstacle and a message of about a possible collision will be displayed on the screen.

Other elements connected to the GPIO socket include signalling and illuminating diodes and an LCD display controlled by the LCM 1602 system performing the function of a converter between the 12C bus and the LCD screen. For communication between the user's computer or smartphone and the test stand, a wireless connection system operating in the WiFi technology is used. For one of the Raspberry PI 2 USB ports, a TL-WN725N network card is connected. The transmission protocol is the classic TCP/IP v. 4 protocol. The use of WiFi transmission in the 801.11n technology simplifies the programming of the robot. The range is several hundred metres in an open area, and the transmission is encoded, which ensures high security. The designed and constructed test stand is universal and enables many hardware and programming solutions to be tested. The test stand is scaled, thus allowing additional devices to be installed on the robot, such as a camera, a sampling system, etc. It enables the analysis of using the Python language for the control of a mobile system to be made.

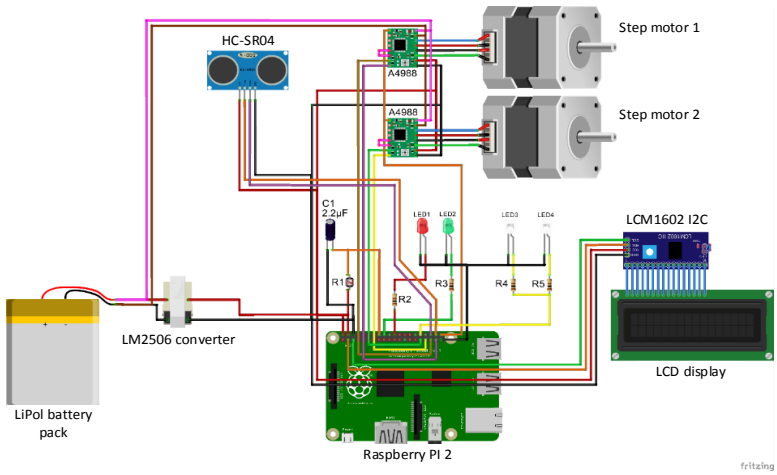

Fig. 3. A schematic diagram of the test stand control system [source: own study]

3. BUILDING THE CONTROL APPLICATION IN THE PYTHON

The Python language was created in 1991 by the Dutch programmer Guido van Rossumen. It is a general-purpose high-level programming language with an extended package of standard libraries. The syntax is distinguished by transparency and conciseness. Python supports various programming paradigms, such as object-oriented, imperative and, to a lesser extent, functional. It has a fully dynamic file system and automatic memory management. It is an interpreted language, therefore it is often used as a script language. To operate correctly, it requires the presence of a special, operating system-dependent interpreter, which analyzes the source code, compiles and then executes it. The Python language interpreters are available practically for all operating systems. Its chief asset is openness – it is being developed by Python Software Foundation as an OpenSource project [14, 17].

On the test stand (mobile robot), the Raspbian Jessie operating system was used, which is based on the Linux system architecture. It is a free distribution optimized for correct running on the Raspberry Pi 2 microcomputer platform. As it is an Open Source distribution, it is possible to interfere in the system's code to adapt it to a specific task. Due to the small size of the robot and its nature (a mobile device), the connection of peripheral devices in a classic manner was impossible. A terminal connection using an SSH protocol stack was used. As a result, a remote, textual and, most importantly, encrypted connection in a client–server mode was obtained. Due to the limitations of the Raspberry PI 2, the control application in the Python language was created within the dedicated IDLE Python version 3.5 programming environment on a traditional PC in the MS Windows 7 environment. The written program was then copied onto the micro-SDHC memory card of the Rasbpberry Pi 2 microcomputer located in the robot using the WinSCP application.

The construction of the test stand and the project assumptions explicitly determined the software structure. The application was divided into two independent, but closely working parts. The first part is software operating on the robot, performing the role of the server, and the second part – client software running on the operator's computer or smartphone (table 2).

Tab. 2. Client and server tasks [source: own study]

Client tasks	Server tasks
• making connection with the server, • running the pygame library, • reading out the information on the pressed key, • matching the command to the operator's reaction • sending the command to the Server, • sending the relevant command in the case of an error or no response.	• the GPIO pins definition, • defining the GPIO pins as an Input/Output, • setting the initial states of the GPIO pins, • reading in the definitions of steps, moving directions, measurement and additionnal illumination, • receiving the command from the Client, • taking the relevant action according to the received command.

3.1. Server software

The server's software is run on the Raspberry Pi 2 microcomputer. Its basic job is to receive the command transferred from the Client and to take relevant actions (Fig. 4). The program 'Server' is run automatically upon the operating system start. The Server connects with the Client and remains in constant contact with it. The communication takes place via the UDP protocol. This is a connectionless protocol of the fourth layer of the OSI/ISO model. This protocol has no mechanisms for the control of the flow and retransmission of packages (other OSI/ISO model layers are involved in the correction of errors). An advantage is a higher speed of data transmission between the Server and the Client. The UDP protocol does not burden the host with additional activities, whereby any delays are very small compared to other protocols [18].

The Server's program has been written in the Python language version 3.5 (2016) using libraries containing the procedures of I/O device handling and standard computer functions operation. The Python language version dedicated to the Raspberry PI 2 includes also libraries specific to microcomputer assemblies (operation of the GPIO connector) [16, 18].

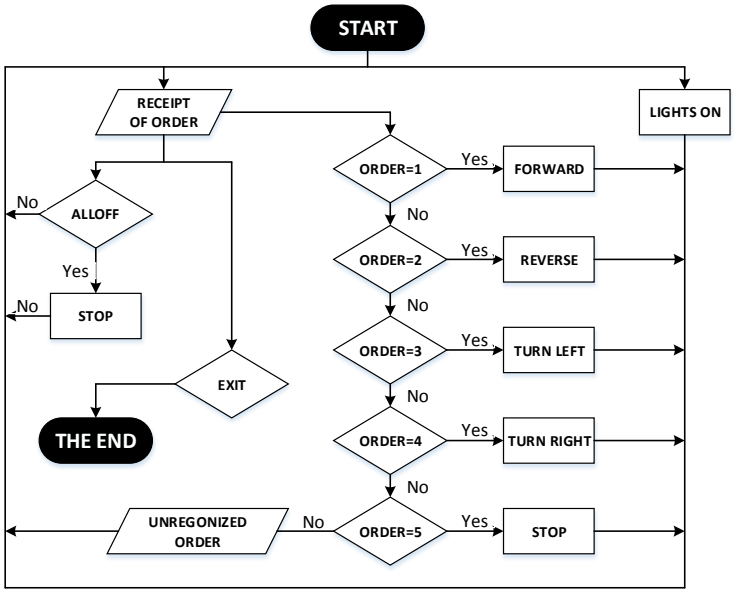

Fig. 4. The block diagram of the application 'Server' [source: own study]

When creating the Server application, the following were used:

- SocketServer – a library responsible for making a connection and creating the so-called connection socket, in accordance with the IP V.4 protocol,
- RPi.GPIO as GPIO – a library for handling Raspberry Pi 2 GPIO terminals, which enables the control of GPIO terminals without giving directly their logic addresses, and the indication of a logic stage (0 or 1) on a specific pin; to define specific pins, respective numbers are used,
- time – a library that enables the use of a clock in the program code, thanks to which it is possible to count the time, e.g., when measuring the distance,
- lcddriver – a library that handles the LCD display and the I2C converter dedicated to its operation; the library enables the addressing of the I2C converter, textual entering the text to be displayed on the display and controlling the backlight of the LCD display,
- socket – another library designed for the operation of the connection socket; it performs the function supplementary to that of the SocketServer library.

The Server takes actions as per the command received from the Client (Fig. 4). These actions include: controlling the stepper motors, measuring the distance from an obstacle, controlling the lighting, etc. The detailed analysis and listing of the Client and Server programs would go far beyond the scope of this paper. Therefore, only the most important fragments of the program will be covered. According to the block diagram (Fig. 4), the programs consist of many procedures

and functions and are divided into blocks, each of them performing a specific task. For example, one of the functions performed by the Server is, in the case of insufficient external lighting, the additional illumination of the space in which it moves. The definition of the instruction light() in the Python language is as follows:

```
def light ():    # definition responsible for measurement
# illumination
    light = 0
    GPIO.setup(photo, GPIO.OUT)
    GPIO.output(photo, False)
    time.sleep(0.0002)
    GPIO.setup(photo, GPIO.IN)
    while (GPIO.input(photo) == GPIO.LOW): light += 1
    if light >= 1000:         GPIO.output(ledb, True)
    elif light < 1000:        GPIO.output(ledb, False)
    print 'Light intensity level: %i' %light
```

The purpose of this instruction is to measure the light intensity using a photoresistor installed on the robot. The Raspberry Pi 2 examines the step response on the pin, to which the photoresistor is connected. The step response is the testing of the system's response to the change in the signal, which is generated on the input connector, from the low to high state. The testing involves the measurement of the time of capacitor charging up to a level of 1.65V (this is the level of the receipt the high state on the input pin). This time is dependent on the resistance that the photoresistor has at the moment when the light falls upon it. The lower the photoresistor resistance, the faster the capacitor charging voltage increases [6]. The Raspberry Pi examines the time between the high and low states. Thereby, the lighting level is defined. It the lighting turns out to be too low, the 'Server' application lights up the working space by passing a signal to the GPIO pins [16,18], to which white LEDs are connected. If the value is greater than or equal to 1000, the pin state will be set as high and the LEDs will light up. If the value is less or equal to 1000, the diodes will go off.

Another task of the Server is to perform the measurement of the distance as it moves forward. The definition of measurement() is as follows:

```
def measurement():     # measurement of distance from obstacle
                       # via HC-SR04 sensor
GPIO.output(trig, True) # sending impulse 10µs in length
time.sleep(0.00001)
GPIO.output(trig, False)
start = time.time()    #start time measurement
while GPIO.input(echo)==0:
```

```
start = time.time() #waiting for echo
while GPIO.input(echo)==1: #stopping time
stop = time.time()

# computing impulse length
impulse = stop-start
#computing distance (34300 sonic speed in air cm/s)
global distance
distance = impulse * 34300
distance = distance / 2
print "Distance from obstacle: %0.1f" %distance
```

The definition of measurement() sends out a 10μs-long impulse to the TRIG pin of the HC-SR04 ultrasonic sensor. The signal generates an ultrasonic wave to be emitted by the sensor. This is followed by a period of waiting for the signal on the ECHO pin, to which the high state is fed as soon as the ultrasonic wave has returned to the sensor. The time measured between the end of feeding the signal onto the TRIG pin and the appearance of the high state on the ECHO pin defines the distance of the obstacle from the sensor. For this purpose, the velocity of sound wave propagation in air should be taken into account, which is approximately 34300 cm/s. To calculate the distance, the time value is multiplied by the velocity of sound propagation in air, that is 34300 cm/s. Thus obtained distance is the path covered by the sound from the sensor to the obstacle and back; therefore, in order to obtain the distance from the obstacle, the result is divided by 2. The of the presented procedures will be the definition of stepper motor operation. There are several such procedures in the program, which are responsible for motor rotation direction (forw(), back(), right(), left() and stop()). The procedures are very similar. The only difference arises from the selection of the required motor rotation direction. For example, the definitionforw():

```
def forw():      # instruction for robot moving forward
    GPIO.output(direction1, True)
    GPIO.output(direction2, False)
    GPIO.output(ledz, True)
    GPIO.output(ledc, False)
    print "\nJade forward ...\n"
    lcd.lcd_clear()
    lcd.lcd_display_string("Go", 1)
    lcd.lcd_display_string("   forward!", 2)
    step()
```

The first item is the definition of the logic state on the direction pins direction1 i direction2. The high logic state at the terminal direction1 means motor rotations in the anti-clockwise direction, while the low logic state on the direction2 pin is responsible for the motor movement in the clockwise direction.

Another item is operation signalling with LEDs; the high state on the ledz pin lights up the green LED, while the low ledc state puts off the red LED. This is followed by clearing the LCD display (lcd.lcd_clear()) and displaying text information on the selected instruction (lcd.lcd_display_string()) on the LDC display. After ending of the signalization, the start of the stepper motors follows – the instruction step():

```
def step():                    # motor step definition
    GPIO.output(en, False)
    motor1.start(1)
    motor2.start(1)
```

The definition step() feeds the low state onto the pin en, whereby the stepper motors are able to rotate. Then, feeding of the stepper motor control signal is started. For controlling the stepper motors, the PWM signal was used, as its generation does not cause any delays associated with maintaining the state of the pins, by specifying appropriate time values in the program. Owing to this solution, no additional delays occur.

For the receipt of the command and its proper definition, the command_receipt class is responsible. This class is also responsible for launching the definition responsible for lighting the working space – the definition light(). The next step is the receipt of the command from the Client. After receiving the command, the Server starts its analysis. In the case of the ‘ALLOFF’ and ‘EXIT’ commands, which are responsible for stopping the motors should the Client program abort, the Raspberry Pi sets the high state on the pin en and lights up the red LED.

Similarly, procedures and functions responsible for the operation of all test stand elements have been created. Written in the Python language, the 'Server' application is versatile and scalable. The program skeleton has been created as split into functional parts, whereby adding further elements to the robot will only require writing in additional procedures to operate the installed items. After verifying and testing, the program was copied onto the Raspberry PI 2 memory card.

3.2. CLIENT SOFTWARE

The objective of the project was to create a universal client application to be run both on a traditional PC and on a smartphone with the Android operating system. The purpose of the "Client" application is to remotely control the mobile

robot by the operator. The direction choice and other commands are assigned to respective keys on the computer's keyboard. In the case of the smartphone application, a simple graphical interface has been built, which features push-buttons for controlling the vehicle. The pygame library utilized in the application allows a joystick to be additionally used, but this has not been implemented in the current program version [3]. A characteristic feature of the Python language is the easy exchangeability of applications written in this language between different operating systems or devices [1, 5, 19]. A single code (text file) is created, and then, by compiling it, the interpreter adapts it to a specific system. This immensely facilitates programming. In the case of the "Client" application, this enabled the creation of a kernel common to both operating systems, after which the program was adjusted to the requirements of the Windows and Android operating systems, based on the shared part of the code.

Tab. 3. Description of keys in the pygame library [15].

K_UP	Key „Arrow Up”
K_DOWN	Key „Arrow Down”
K_LEFT	Key „Arrow Left”
K_RIGHT	Key „Arrow Right”
K_ESCAPE	Key ESC

The analysis of the block diagram (Fig. 5) and table 2 shows that the basic job of the 'Client' application is to make a connection with the Server and send to the Server the codes of the commands responsible for performing specific tasks by the robot. Before starting to create the software, a decision was made to write the entire application from scratch. This applied especially to key operation procedures. It turned out, however, that the Python language environment included a ready-made library, called pygame, that was used by game producers. The examination of the library showed that it lent itself perfectly to the 'Client' application being designed. Therefore, it was decided to abandon the original concept and use the ready-made solution. Thanks to the library, the codes of keys pressed by the operator are red out. Key descriptions incorporated in the library (table 3) make it possible to determine, which key has been pressed or released.

For controlling the robot, 5 keys are sufficient: the arrowup, arrowdown, arrow left, arrow right and the ESC keys. The arrow keys are responsible for respective model movements: forward, backward, to the right and to the left. The ESC key terminates the software operation.

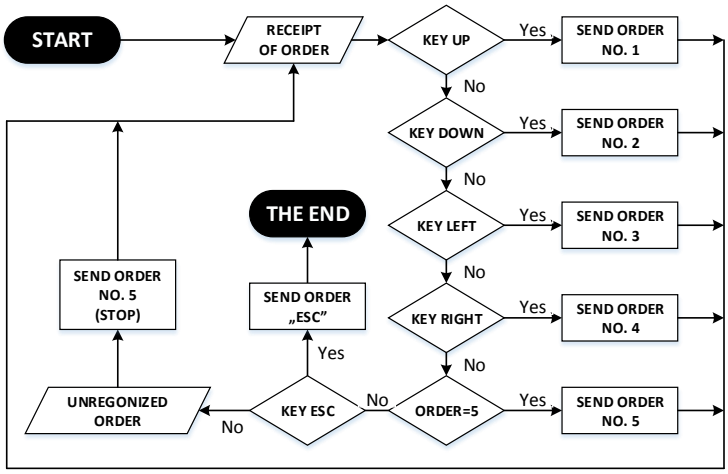

Fig. 5. The block diagram of the application 'Client' [source: own study].

Similarly, as the 'Server' program, the client application requires the declaration of libraries to be used in the further part of the code to be made in the program's heading. Moreover, the Server's IP address and the number of the communication port required for connection have been defined:

```
import socket
import time
import pygame
broadcastIP = '192.168.2.1' # IP server address
broadcastPort = 6666    # Communication port address
```

To ensure the smooth operation of the robot, the remote control requires the smallest possible user's response delay. Therefore, the interval within which the user's actions are interpreted and automatically sent to the Server has been set. The time interval is, at the same time, adjusted to the operations to be executed by the Server between the receipt of the commands. Then, the connection socket is created and opened:

```
interval = 0.2  # Command sending interval - 0.2s
regularUpdate = True
sender = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,
socket.IPPROTO_UDP)
# Creating socket (socket)
sender.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1
sender.bind(('0.0.0.0', 0))
```

The next steps include the initiation of the declared variables, launching the pygame library and assigning instructions to respective keys.

```
hadEvent = True
moveUp = False
moveDown = False
moveLeft = False
moveRight = False
moveQuit = False
pygame.init()

screen = pygame.display.set_mode([300,300])
pygame.display.set_caption("Robot - Operator")

def operations(events):
global hadEvent
global moveUp
global moveDown
global moveLeft
global moveRight
global moveQuit

for event in events:
# Setting variables after aborting by the user
if event.type == pygame.QUIT:
hadEvent = True
moveQuit = True
,


elif event.type == pygame.KEYDOWN:  hadEvent = True
if event.key == pygame.K_UP:        moveUp = True
elif event.key == pygame.K_DOWN:    moveDown = True
(…)
if event.key == pygame.K_UP:        moveUp = False
elif event.key == pygame.K_DOWN:    moveDown = False
(…)
```

After configuring the libraries and key actions, starting of the program's main loop follows. Specific commands are assigned to respective key actions: 1, 2, 3, 4 and 5 are assigned to the Server's actions.

```
try:      print 'Press [ESC] to exit ...'
# Main loop
while True:
```

```
operations(pygame.event.get())
if hadEvent or regularUpdate:
hadEvent = False
driveCommands = ['X']

if moveQuit:            break
elif moveLeft:          driveCommands = '3'
elif moveRight:         driveCommands = '4'
elif moveUp:            driveCommands = '1'
elif moveDown:          driveCommands = '2'
else:
driveCommands = '5' # Sending command 5 - STOP
```

After assigning a specific command, sending it to the Server follows via the connection made previously.

```
sender.sendto(driveCommands, (broadcastIP, broadcastPort))
time.sleep(interval) # Waiting according to interval
sender.sendto('ALLOFF', (broadcastIP, broadcastPort))
```

In line with the employed programming method, the 'Client' program has a modular construction and is written in a universal manner. Similarly, as for the second part of the application, 'Server', its further expansion is possible by writing in successive procedures and functions (e.g. the use of a joystick for controlling the mobile robot's movement).

4. CONCLUSIONS

The developed test stand became a basis for carrying out the analysis of the use of the Python language for the control and management of the operation of an autonomous track-laying mobile robot. The structure and principles of programming in the Python language are similar to those of other programming languages. It enables structural and object-oriented programming, and its syntax is modelled on the C and C++ languages. Its advantage is the cross-platform capability and a huge library of extensions, created by independent programmers. The best examples are the pygame libraries used in the design, created originally to operate keyboards and joysticks in games, and the specialized library for operating the GPIO connector of the Raspberry PI 2 platform. Creating the user's own procedures, functions and definitions is straightforward, and the created code is legible and easy to analyze. The tests of the robot have confirmed the correctness of the programming algorithms used. The robot moves smoothly

and correctly analyzes the environment, in which it moves [20]. The greatest disadvantage and, on the other hand, advantage is the fact that the Python language is an interpreted language. This translates into the size of the code, the requirement for the presence of an operating system and a dedicated compiler, and a lower operation speed compared to applications created using the C or C++ language (compiled languages). In spite of this drawback, the analysis has shown the Python language to be useful for programming mobile robot control systems, especially for users having no extensive programming knowledge [1, 4, 8, 9, 10]. The best evidence for the capability of the Python language is its use by the NASA for controlling the launch of space shuttles and in the Nebula project [15].

REFERENCES

[1] GUERRA H., CARDOSO A., SOUSA V.: *Remote Experiments as an Asset for Learning Programming in Python*. International Journal of Online Engineering, 12(4), p. 71–73.
[2] KARVINEN K., KARVINEN T.: *Czujniki dla początkujących* (*Sensors for beginners*). Helion, Gliwice, 2015.
[3] LEWIS A. J., CAMPBELL M., STAVROULAKIS P.: *Performance evaluation of a cheap, open source, digital environmental monitor based on the Raspberry Pi*. MEASUREMENT, 87, 2016, p. 228–235.
[4] MATVEEV A. S., HOY M. C., SAVKIN A. V.: *Extremum Seeking Navigation Without Derivative Estimation of a Mobile Robot in a Dynamic Environmental Field*. IEEE Tranactions on control systems technology, 24(3), 2016, pp. 1084–1091.
[5] MONK S.: *Raspberry Pi. Przewodnik dla programistów Pythona (A Guide for Python Programmers)*. HELION, Gliwice, 2013.
[6] MONK S.: *Raspberry Pi. Receptury (Recipes)*. Helion, Gliwice, 2014.
[7] PARK J.-H., YANG H.-S., LEE J.-H.: *Remote Power Control System using the Raspberry Pi*. The International Journal of Advanced Smart Convergence, 4(2), 2015, p. 120–123.
[8] ROBINSON A., COOK M.: *Raspberry Pi. Najlepsze projekty (The best designs)*. Helion, Gliwice, 2014.
[9] SOBASZEK Ł., GOLA A., VARGA J.: *Virtual Designing of Robotic Workstations.* Applied Mechanics and Materials, 844, 2016, p. 31–37.
[10] SOBASZEK Ł., GOLA A.: *Perspective and methods of human-industrial robots cooperation.* Applied Mechanics and Materials, 791, 2016, p. 178–183.
[11] http://www.af.mil, U.S. Air Force, Photos, 2016.
[12] http://www.forum.arduino.cc – Forum devoted to the microcomputer subject matter. Descriptions of using the elements utilized for building the model.
[13] http://www.majsterkowo.pl/rapsberry-pi-pierwsze-kroki, 2015.
[14] http://www.python.rk.edu.pl/w/p/python-co-jest-i-do-czego-mozna-go-uzyc, 2015.
[15] http://www.raspberry-pi-geek.com – Site devoted to the Raspberry Pi microcomputer, 2016.
[16] http://www.raspberry-pi-geek.com/howto/GPIO-Pinout-Rasp-Pi-1-Model-B-Rasp-Pi-2-Model-B, 2016.
[17] http://www.wikipedia.pl – Information used in theoretical descriptions, 2016.
[18] https://www.osamajaber.wordpress.com/2014/04/15/rpis-gpios-over-udp, 2015.
[19] https://www.pl.python.org/docs/ref/node1.html, 2016.
[20] https://www.raspberrypi.org – Manufacturer of Raspberry Pi, 2016.