Development of a manipulator control system via the Internet

A V Pilipenko, A S Vetrov, A Y Pilipenko, N V Goncharova
OGU named after I.S. Turgenev, Oryol, Russia
E-mail: a@pilipenko.info

Abstract. In this paper, the authors examine in detail the technology of developing an IoT project based on the AVR Atmega328 microcontroller, provide a detailed description of the system code and approbation of the results, as well as the authors touch on topical and important issues on creating a web server on Arduino and how it interacts with the client.

1. Introduction
In the world, information technologies are developing every day and the number of “connected” devices is growing, and with it the number of examples of using the Internet of Things (Internet of Things, IoT) in the economy, energy, industry, housing and utilities, agriculture, transport, health care and others [1]. In foreign practice, there are successful examples of IoT implementation at the initiative of both the government and business. For example, with the support of the state in the countries of the European Union, South Korea, China and India, smart city technologies are being introduced, which make it possible to increase the efficiency of energy management and traffic flows. Industrial IoT technologies underlie Industry 4.0: according to scientists, their implementation will increase the productivity of industrial enterprises by 30% until 2025. In this regard, the implementation of research on the concept of IoT and Industry 4.0 elements is necessary for more intensive development of technological processes.

In this paper, the authors solve the problem of controlling the servos of the Robot Arm Kit using the Ethernet interface, avr microcontroller and the IoT platform. To achieve the goal, the following tasks were performed: analysis of decisions in the field; development of the block diagram of the designed device; development of the electrical circuit of the device being developed; development of the algorithm in the form of a flowchart (block diagram); code development for the control system.

2. Concept
During the study of information on this topic, it was decided to use the Arduino UNO board in conjunction with the corresponding Ethernet Shield [2]. Arduino Uno controller is built on the ATmega328 [3]. The platform has 14 digital inputs / outputs (6 of which can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connector, a power connector, an ICSP connector and a reset button.

In the Robot Arm Kit manipulator used, mobility is provided by precision servo drives of two types: DF15MG and DF05BB [4]. These two types of servo drives have significant differences in geometrical dimensions, due to which the first servo drive has twice the torque and angle of rotation. Servomotors use for their performance voltage in the range from 4.8V to 5.7V, so the power will be connected to the servos directly from the board. Since it is necessary to interact with 6 servos, they will connect to 3, 4, 5, 6, 7 and 8 discrete outputs on the board, they will be controlled by the standard library Servo.h (Figure 1). While studying the possible implementations of the software component of this project, it
was decided to use Arduino, with the Atmel mega328 AVR microcontroller, as the “server” of the receiving and processing POST and GET requests, via the Ethernet2.h library. The answer to the GET request will be the html code of the page. POST requests will be processed and controlled by servos, depending on the information contained in them.

Figure 1. Electrical wiring diagram of servos.

The structural diagram of the project under consideration is presented in Figure 2. This structural diagram shows the interaction of the system elements with each other, namely: “Client” is the system element that sends requests via TCP / IP to the server (the device being designed); Ethernet Shield - an element of the system mediating data transfer between the client and the server; “Server” is an element of the system responsible for processing information and the subsequent management of the elements of the system; Robot Arm Kit is an element of the system that needs to be controlled depending on the information received from the “client” [5,6].

Figure 2. Block diagram.

Arduino is an open-source platform that consists of two main parts: the board itself (often called a microcontroller) and software (a special shell for programming the board) or IDE (Integrated Development Environment) [7]. The software runs on a personal computer and allows you to write the developed code to the board.
3. Software development

Requests for Arduino UNO will be processed through the library (Ethernet2.h). After connecting the device to the network, it will be ready to receive and process requests. To work correctly, you must open the browser and in the address. To work correctly, you need to open the browser and enter the device’s ip address in the address bar, after which the browser (the client) will form a socket with a GET request and send it to the server (device), after which the html code of the page will be sent in response. Two signed text fields and a button that will be located in the iframe, when you fill in the fields and press the button, a POST request will be sent to the server, which will contain data from the fields. When a request is received, the server will determine its type (POST or GET), and in accordance with this, it will either manage the servos, depending on the data received, or respond to the page's html code.

When receiving data on the need to control the servo drives, voltage will be supplied to the corresponding leg of the Arduino UNO via the standard library (Servo.h). The processing of the information obtained in the requests is performed by cycling through the elements, and sorting them in the same way.

First, libraries, variables, etc. are initialized. Library connection: #include <SPI.h> is a library that allows the use of serial synchronous data transfer protocol used by microcontrollers for data exchange. #include <Servo.h> is a library for managing servos. #include <Ethernet2.h> is a library for processing requests. Declare strings for intermediate storage of information from a query in the program memory:

String tmp_string = String (); String query = String (); String query_get = String (); String query_post = String (); String query_method = String (). Servo Announcement: Servo servo1; Servo servo2; Servo servo3; Servo servo4; Servo servo5; Servo servo6. Declaration of variables for information processing:

int j; int i = 0; int serv; int angle. Setting server settings: IPAddress ip (192, 168, 1, 177); byte subnet [] = {255, 255, 255, 0}; byte mac [] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED}. Setting the server port 80: Ethernet Server (80).

The following is the contents of the Setup function, in which the server and servo drives are initialized once: Ethernet.begin (mac, ip, subnet); server.begin (); Serial.print ("server is at "); Serial.println (Ethernet.localIP ()); servo1.attach (2); servo2.attach (3); servo3.attach (4); servo4.attach (5); servo5.attach (6); servo6.attach (7).

The contents of the infinite loop will be divided into functional blocks. Getting information from the client: EthernetClient client = server.available (). Checking the relevance of the information: if (client). Preparing variables for storing information: j = 0; query = ""; query_get = ""; query_post = ""; query_method = ""; boolean CLB = true. The loop works as long as the client is connected: while (client.connected()). The loop works as long as there is raw data: while (client.available ()). Gathering melon requests: char c = client.read (); query = query + c. Checking the end of the request: if (c = = ’
’ && CLB). If the request is not completed, then a cycle is started, working as long as there is POST data from the while (client.available ()) request, in which the POST data is saved: char b = client.read (); query_post = query_post + b. Next, using the for loop, the incoming data is sorted by the permutation method. After that, the end of the else if request (c = = "\n") is re-checked, if the result is true, then the value in the variable CLB = true changes, if it is false, then an additional check is performed else if (c! = "\R"). And when obtaining a logical unit, the reverse change of the value CLB = false is performed.

The initial values of the variables responsible for the servo drive number and its rotation angle are set (serv = 0; angle = 0). Then the code for checking the status of the request is written (if (query_post.indexOf ("serv")! = -1 & & query_post.indexOf ("&"))! = -1 & & query_post.indexOf ("serv") <query_post.indexOf ("& ")) and if a logical unit is received, then information is obtained about which servo drive must be controlled: j = 0; tmp_string = ""; for (i = query_post.indexOf ("serv"); i <query_post.indexOf ("&"); i ++) {if (j == 1) {tmp_string = tmp_string + query_post [i];} if (query_post [i ] == "=") {j = 1;} serv = tmp_string.toInt (). Now you need to get information about the angle of rotation of the servomotor, with the received index. The status of the request to change the angle if (query_post.indexOf ("angle")! = -1) is checked and, when a logical unit is received, the values of the angle are obtained: j = 0; tmp_string = ""; for (i = query_post.indexOf ("angle"); i
<query_post.length (); i ++) {if (j == 1) {tmp_string = tmp_string + query_post [i]; } if (query_post [i] == '=' ) {j = 1; } } angle = tmp_string.toInt ().

When the initial parameters are configured, the html page is created for the user interface and the progressive html code is sent to the client: client.println("HTTP / 1.1 200 OK"); client.println("Content-Type: text / html; charset = utf-8"); client.println();

When the initial parameters are configured, the html page is created for the user interface and the progressive html code is sent to the client: client.println("HTTP / 1.1 200 OK"); client.println("Content-Type: text / html; charset = utf-8"); client.println(); client.println("<! DOCTYPE html>"); client.println("<html lang = " ru">" ); client.println("<head>"); client.println("<meta charset = " UTF-8">" ); client.println("<title> Servo </ title>" ); client.println("</ head>"); client.println("<body>"); client.println("<form id = "PostForm" method = "post" action = "192.168.1.177">" ); client.println("<p> <b> Servo: </ b> <br>"); client.println("<input name = "serv" type = " text" size = " 20">" ); client.println("<p> <b> Angle: </ b> <br>"); client.println("<input name = "angle" type = " text" size = " 20"> "<br>" ); client.println("<input type = "submit" value = " Send"/>"); client.println("</ form>" ); client.println("</ body>" ); client.println("</ html>" ).

It remains to consider the classic servo control algorithm, in which the state of variables with the servo drive number and rotation angle (if (serv! = 0 and angle! = 0)) is checked if the value has changed, if the value has changed, then the servo motor will be completely searched according to the following principle: if (serv == 1); else if (serv == 2); else if (serv == 3). If the state of the variable coincides with the index, for example, with the number 1, this engine is rotated to the specified angle: servo1.write (angle). After the transfer is completed, it is correct to pause the exchange of information with the client (client.stop ()).

The algorithm of the system in the form of a flowchart is presented in Figures 3-4 and describes the following functions: 1 - Start; 2 - check for power; 3 - The end of the program; 4 - getting information about the client connection; 5 - check for the presence of a client; 6 - resetting variables for storing and processing request information; 7 - check client connection; 8 - cyclic check of connection availability; 9 - character read request data; 10, 11, 12, 13, 14 - check for the end of the request header; 15 - check the availability of data of the request body; 16 - read the request body data; 17, 18, 20, 21, 22, 23, 24, 25 - parse the request header and get data about the type of request; 26 - the beginning of the formation of a response to the client; 27 - checking the presence in the request body data on the servo number that needs to be controlled; 28, 29, 30, 31, 32, 33, 34 — retrieving data from the request body for the servo number; 35 - checking the presence in the request body of the angle data on which the servo needs to be turned; 35, 36, 37, 38, 39, 40, 41, 42 — retrieving data from the angle request body; 43 - completion of the formation of a response to the client; 44 - checking the availability of information about the servo number and the angle at which it must be rotated; 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56 — select the appropriate servo number and rotate it to the required angle; 57 - closing the connection with the client.
Figure 3. First part of the algorithm.
4. Results
As a result of testing, the functionality of the device was confirmed through repeated testing and a “screenshot” of its work was made, which is presented below in Figure 5. Figure 6 shows an image of the assembled device.
This “screenshot” of the program shows that when you send a GET request to the server on the Arduino UNO, the “client” (in this case, the browser) receives the html page code in response. On which you can see 2 fields to fill in and the button to send this data POST request to the server, with subsequent processing of this data [8].
During the creation of this project, there were problems displaying the html code page sent by the server, most likely they occurred because the device was tested without connecting the client to the “clean” Internet (without a proxy server). She managed to solve when using the browser "Opera".

In this paper, the code is described in full, but without using operator brackets. After the successful assembly, 30 requests were sent to the server with various servo control data, all of which were processed correctly by the system.

References
[1] Kupriyanovsky V.P., Namiot D.E., Sinyagov S.A. 2016 International Journal of Open Information Technologies 4 (2)
[2] D’Ausilio A. 2012 Behavior research methods 44(2) 305–313
[3] Barrett S. F. 2010 Arduino Microcontroller Processing for Everyone (Morgan and Claypool Publishers: San Rafael)
[4] Kazanzides P., Chen Z., Deguet A., Fischer G. S., Taylor R. H., DiMaio S. P. 2014 IEEE International Conference on In Robotics and Automation (ICRA) 6434–6439
[5] Tristo G. et al. 2015 The International Journal of Advanced Manufacturing Technology 78 (9-12) 1511–1521
[6] Murdoch S. J., Stephen L. 2005 Embedding covert channels into TCP/IP (Springer: Berlin, Heidelberg)
[7] D’Ausilio A. 2012 Behavior research methods 44(2) 305–313
[8] Ramadas M., Shawn O., Brett T. 2003 Detecting anomalous network traffic with self-organizing maps (Springer: Berlin, Heidelberg)