Design and Implement a 6DOF Anthropomorphic Robotic Structure

Ovidiu Stan and Andrei Manea

***Abstract*—In recent years robots have begun to become an active presence in everyday life in various forms. Helpful robots that offer help and support to people with special needs or social robots that are able to interact with customers in stores. The presented paper presents a method for design and implement a 6DOF robotic arm over the IoT concept. The core of this project is a Raspberry Pi board and the I2C protocol.**

***Index Terms*—DOF, anthropomorphic robotic, raspberry PI.**

I. Introduction

For years robots have been slowly making their way into every aspect of our lives from huge scale industrial processes to every day around the house appliances and small gadgets. One of the main reasons that robots have gained all this popularity is in part that they can help with small tasks around the house for example automated vacuum cleaners, lights windows and so on, they make lives easier and public perception slowly changed for the better. The problem is that the public doesn't realize how much more robots do for us, they do the work no human wants or can do. They handle materials and substances that are highly dangerous [1].

With the emergence of IoT, Cloud-Computing, humanoid robots and intelligent robot services, more and more studies have been developed on robot control and on how robotics have intersected with IoT [2], [3]. Two aspects need to be considered for robot control. The first concerns the limitation of resources managed by the robot's system due to large volume data constraints and costs [3]. The second aspect concerns the cost of integrating robots with IoT.

II. Core Concept

A. 3D Printing and Material Choice

3D printing, also known as additive manufacturing [4], [5] is an innovative technology which allows fast prototyping. Its working principle is based on layer by layer composition of an object. All layers are pressed onto each other at a distance which defines the resolution of the 3D printing process. The smaller the distance, the better the resolution and overall quality of the final product.

Regarding the 3D printed parts, ABS [6] was chosen because of its mechanical properties: flexibility, toughness and improved impact resistance and since it is used in a different field [10]. The only downside was the difficulty of obtaining the required bed temperature (110° C) and extruder temperature (230° C).

B. 6-DOF Structure

Robotic arms are a common approach designed for resolving automated tasks. One of the most common structures, yet a relatively complex one, is the six degrees of freedom structure. It resembles a closer representation of human arm, being considered an anthropomorphic structure [7], [7].

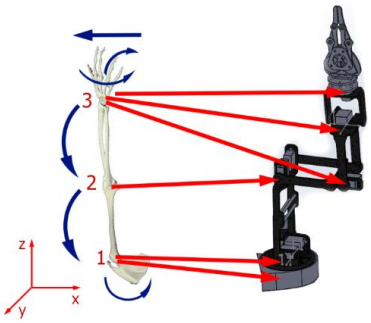


Fig. 1. Human arm - 6 DOF structure.

In order to reproduce the mechanical capabilities of a human arm, one must analyze the physiological and kinematic properties of interest. Figure 1 presents a simplified analogy made when adapting the 6-DOF structure for the current project purpose used in order to notice the fact that are four positioning and two orientation actions.

C. Finite State Machine (FSM)

One of the most used tools for conceptualizing models are finite state machines, they represent the interactions at a discrete level in the system [9]. FSM represents how an activity can change its behavior over time reacting to internal or external triggering events.

In Figure 2 the state machine diagram for the system is shown. After the system is powered, the current sensors start measuring the current drawn by the servo motors. The data gathered is sent to the Arduino slaves, they have had uploaded onto them the source code which takes the current values as an input through the I2C bus that sends them to the master controller which displays the data. Two of the Arduino NANO boards are tasked with monitoring the current, the third one is tasked with controlling the stepper motor and managing the data from the reflectance sensor. The same figure also applies to the third slave module, their design in finite state machines is identical. After the system is initialized, the sensors start sending data to the slave, at the same time the stepper starts moving, when encountering the white strip on the base of the robotic arm, the reflectance sensor sends a signal to the slave and the stepper stops in that position.

Manuscript received March 29, 2019; revised May 2, 2019. The research presented in this paper was supported by the following projects: ROBIN (PN-III-P1-1.2-PCCDI-2017- 0734) and SeMed (2933/55/GNaC 2018)

Ovidiu Stan and Andrei Manea are with the Technical University of Cluj Napoca, Faculty of Automation and Computer Science, Department of Automation, Cluj Napoca, Romania (e-mail: ovidiu.stan@aut.utcluj.ro).

DOI: 10.7763/IJMO.2019.V9.736

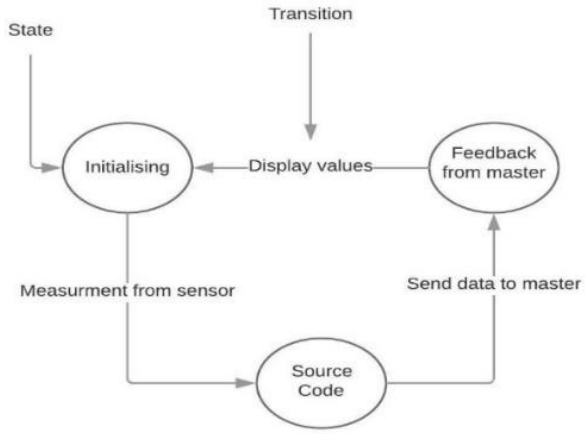


Fig. 2. Finite state machine.

III. Control System

A. Electrical Design

The focus of this section is the design of the electronics of the robotic arm and the tools used for this stage of the development. The electronical assembly should be able to deliver power to the transmission, to the motors, and it should be able the power the electronic circuits and microcontrollers used in the assembly. The 600 watts power supply has two 25 Amps channels, each channel having three outlets for the 3.3 volts, 5 volts and 12 volts, each of them being used fed into a distribution board with outputs for each of the necessary prototyping boards, each receiving the necessary voltage. The wireless device which can control the robotic arm will be powered by a 2S Li-Po battery that provides 8.4 volts and can provide up to 30 Amps of current at peak performance for a time of 10 seconds. In the case of this device that amount of current is not necessary, but the option of a Li-Po battery is preferred because of its versatility over other batteries such as nickel-cadmium, which provides a much smaller output voltage and current. The number of cells is enough for the task at hand, being a Raspberry Pi 0 W, a display and the circuitry required to run everything in the device.

The methodology used when designing the system was straight forward, making the circuit as simple as possible so that if complications arise, they can be easily identified and resolved [10]. Since some of the electronics are done by hand some of the parts in the system might be more susceptible to wearing out faster than others, so we came up with a solution, making the design modular. The system is spread out onto four boards and three layers. This way, if something goes wrong and needs changing, it can be done so in a manner of minutes. One of the most important aspects when designing a circuit, besides the functionality, is to make it as legible as possible for someone that has not previously seen the circuit diagram to be able to understand it. If one has the desire to recreate the circuit, they must be able to do this based on the given information.

As shown in Figure 3, the circuit is designed around the I2C protocol. To make the circuit as simple as possible, wiring wise, the Raspberry Pi master is placed in such a manner as to accommodate all the signals coming from the Arduino Slaves and the servomotor driver coming to the Serial Data (SDA) and Serial Clock (SCL). When taking a closer look at the diagram one can see the signals linked to the SDA and SCL are bi-directional, this is because the I2C protocol is omnidirectional, otherwise known as full duplex, receiving and sending data at the same time. The diagram was constructed in the same manner as the circuit, following the figure from top to bottom, the circuit can be easily reconstrued. Starting from the top of the diagram, the 9 nine step-down modules, LM2596, which receive 12 volts from the power supply and they output 6.2 volts to a board containing 1000 µF capacitors, they are used for attenuating the current spikes generated by the servo motors when they start receiving power, all nine modules are connected to a capacitor. The current sensors, ACS712, are each connected to one of the servo motors in the same fashion as the step-down modules, each sensor is powered by the 5-volt power line. Each of the eight motors receive 6.2 volts from the step-down modules and the sensors measure how much current they draw. There are nine current sensors and nine modules, but they are connected to eight motors, this is because of two reasons. The first one is safety related, there are more modules because if one of them failed, they can be swapped on the spot, the ninth module is not connected to any motor. The second reason is structural integrity, the modules are placed in a tower like formation, each tower consisting of three modules, if each column has the same number of step-downs it makes the whole ensemble more stable, as a bonus they also look more esthetically pleasing. The same reasons apply to the current sensors, the difference being they are not laid out in a tower of three formation, they are positioned one alongside another on the board. The servo motors, MG996R, tasked with the movement of the robotic arm receive the command, PWM, from the servo driver, PCA9685, the command received consists of the position desired for the servomotor. The command is received from the I2C bus. The power necessary for the driver to function is 3.3 volts, supplied by the corresponding line from the power supply.

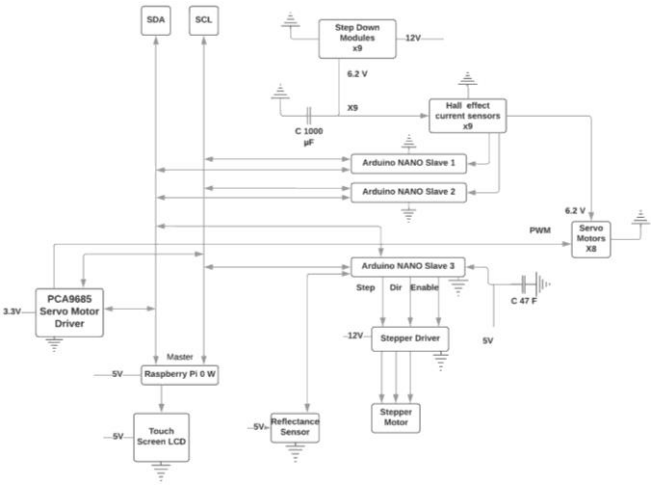


Fig. 3. Electrical diagram.

As stated earlier, the circuit design is centered around the I2C protocol, as it can be easily deduced from the diagram. The positioning of the devices is purposefully in this manner because the Raspberry Pi is the master and, on its SDA, and SCL pins are the bus where all the information is exchanged.

Starting with the essential component, the master microcontroller, it must be supplied with 5 volts in order to function correctly. The master controller sends on the I2C bus the necessary information for the devices, that information ultimately becomes the movement of the arm. The master has three slave devices, three Arduino NANO boards, and their tasks are split up into two. The first two are tasked with sending and receiving measurement from the current sensors. Many pins are required for this, 18 to be exact, and because using a shield or a number ADC, it is best to use two Arduino Nano instead, for two reasons, size and effectiveness. The two slaves receive the current measurement from the sensors connected to the motors and send them via the bus to the master. Each of the boards is connected to the 5 volts power line. The reading of the values from the sensors is done on the slaves which communicates the information to the master. The third Arduino Slave is tasked with the control of the stepper motor. Between the power supply wire and the third slave is a 47 µF capacitor tasked with keeping in check fluctuations in voltage from the power supply. The slave sends the direction, the steps and enables the pins to the stepper driver, A4988. The driver has a passthrough for voltage, it takes as an input from the power supply the 12-volt line and feeds it to the stepper motor, this connection is made through the VMOT pin. The power for the driver itself is handled by the slave, suppling the driver with 5 volts using the VDD pin. The driver sends signals through the 1A,1B and 2A, 2B pins, each pair of pins corresponds to a coil of the stepper motor, polarizing each coil one at a time to move the rotor of the motor. The module features pins for adjusting micro stepping, the pins in question are MS1, MS2 and MS3. Also connected to the third slave is the reflectance sensor, QTR-1A, the sensor data is sent to the Arduino which transmits it to the master. The sensor is supplied via the 5 volts line. Finally, the diagram ends with the touch screen LCD screen connected via a HDMI cable with a micro-HDMI adapter to the Raspberry Pi. Power wise, the screen is supplied by the 5 volts line. It goes without saying that each of the devices are connected to their respective common earth. It is also important to mention what kind of wires were used, for the most important parts of the circuit, where the most power flows, like the supply for the motors and the transition boards for the power supply wires and the capacitors board, 22 AWG copper wires were used, with a resistance of 50 mΩ/m. The rest of the wiring was done using a combination of 26 AWG copper wires, with a resistance of 125 mΩ/m, standard jumper cables, and 28 copper AWG, with a resistance of 200 mΩ/m. When choosing wires, one must take into consideration the length, because it influences resistance but in our case the resistance of the cables is negligible, and how much amps of current they can support, the lower the AWG values the higher the load they can support. In this case the thickest wire used is a 22, with a diameter of 0.64 mm, keeping in mind that the diameter can slightly change if a large amount of current is passed through, creating heat dissipation. The maximum amount of current that can pass through the selected wires is 3.5 amperes.

The Diagram was designed in such a way that it perfectly represents the circuit and it was kept as barebones as possible so that is it easily understandable to the reader. The circuit was designed in such a way that hardware and electrical problems are avoided from the start, problems like heat dissipation, wire insulation and magnetic field cancelation. The wires and devices are arranged in such a manner that the connections between do not create a magnetic field to affect other devices. This is achieved by giving enough space for all the components, but not too much because that would defeat the purpose of using small modules and the wires are intertwined in a way that even if the field is somehow created, it will be nullified.

The wireless device (Fig. 4) is tasked with controlling the robotic arm, controlling its servomotors and stepper wirelessly. The device is incased in a 3D printed casing, made from ABS, which houses a Raspberry Pi 0 W, master, the same model is presented as the master controller in the electrical system. The device is designed in such a way as to be small and compact; its base is a square with 11 cm sides and the corners are polished to be softer to the touch when holding the device. The height of the device is 3 cm, enough space for all the electronics to fit inside the casing with ease. On top of the device a 20 by 4 LCD display is placed which is connected to the Raspberry Pi. Inside of the casing an Arduino NANO is present, as a slave.

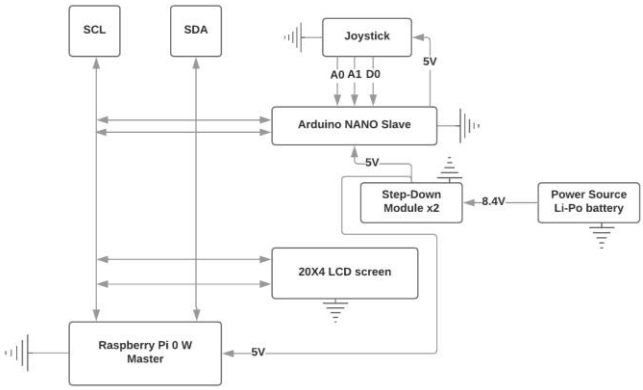


Fig. 4. Wireless device circuit.

As shown in the figure above, a joystick is connected to the slave, it is not directly connected to the master because is it harder to properly interface the joystick directly. So, the solution is using the protocol used previously for the electrical circuit system, the I2C protocol. In the same fashion as before, the SCL and SDA pins of the master are connected to the corresponding pins on the Arduino. The joystick is connected to the Arduino which sends the collected data to the I2C bus. As presented in the diagram (Figure 9.a), the joystick is connected to the slave using four pins, one pin if for power receiving 5 volts from the Arduino, one pin is digital for when the joystick is pressed, and two analog pins are tasked with sending the measurements from the OX and respectively OY axis. In the same manner, the LCD screen is connected via the I2C to the Raspberry Pi master. The power source for the device is lithium-polymer battery 2S, with an output voltage of 8.4 volts. This value is way too high for the master and the slave, they require 5 volts. Instead of using modules, because the space is limited, the solution is using a LM7805 voltage regulator, this regulator has three pins, input, output and ground, they are placed in the circuit in the same manner as before. This issue was present before in the last circuit and the solution is the same,

between the source and the devices, these regulators are placed to get the current from 8.4 volts to an acceptable 5 volts.

B. Node.js Server Architecture

The server is operating on event driven programming principle, which stands for the fact that the control flow state and actions are determined by triggering specific events. The Express HTTP server implementation provides a clean structure for handling incoming HTTP request, facilitating the way one can provide complex data structures based on low level data readings. There is one main module which handles logic operation concluding in the actuators' movement. The TCP/IP port listening is handled in a separate controller, proofing the whole system against deadlocks, handling on request and on receive events for each port opening and manipulating global state variables.

The services layer handles fetching and transmission operations between the whole system and the embedded modules, accounting the information in a local log file. It provides data parsing from byte level to JSON, deploying asynchronous operations which do not interfere with the normal flow. The asynchronous operations' completion emerges back to the main thread through standardized callback functions which deliver either error messages regarding process failure, or the result of the process execution. Using this kind of approach one can sustain a full feedback communication between the server and the embedded system, but due to hardware design limitations, it will not be implemented.

A detailed structure of main Node.js modules used in the application and the relations between them is presented in Figure 5. All the main modules are being called by the App.js file which consist of the core of the server's implementation.

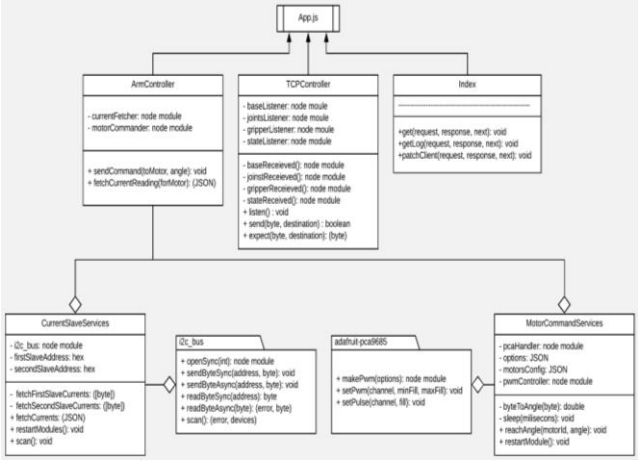


Fig. 5. Node.js server module diagram.

The REST services are enabled for the server in order to provide data fetching from the embedded system. The HTTP server powered by "http" node module is listening on port 3000, in case of production environment deployment. The express server handles URL routing and incoming requests.

One chose to implement only the base route '/' for the local server. All passed data through HTTP is parsed using "body-parser" and "cookie-parser" modules, which delivers JSON data regarding the request components (headers, query parameters, cookies). All server errors were implemented based on standard error handling guidelines. The frontend part of the application is based on Embedded JavaScript, but the only resource which responds with HTML frontend is "not found" server error.

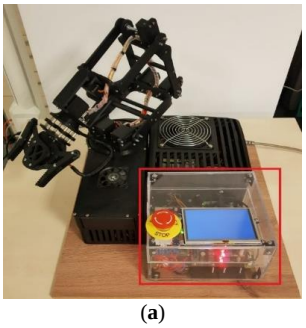
The requests the server responds to are: GET request on base route '/', which checks the authentication token of the requesting client, delivering the latest current readings alongside with the timestamp of the server, in a JSON structured format. GET request on route '/report' returns a JSON file which contains all server log for the current session, after confirming the authentication token. The PATCH on route '/' confirms the end of a session from the WPF application point of view, setting the robot in idle state and deactivating all motor command modules. The Node.js application is set to deploy at the launch of the system, using a custom Bash script.

Considering the application approach for local data transmission and control, once can implement TCP/IP socket communication. Using "net" node module, one can easily configure a listening and responding socket in JavaScript. One must implement the on receive callback method in which all the data received is parsed from byte array to long formats and, depending on its identifier, it is then passed to the proper controller method in order to execute the commands.

Despite of common good practice of handling multiple incoming connections on same socket using a multi-threading approach, one chooses single thread implementation and multiple socket connections deployment. This was done in order to keep consistent local control, the commanding server being able to listen to one data transmitting client at a time, therefore implementing a first come first served principle. This allows better data handling without overcrowding TCP sockets, and being a local application, it is taken as a confident manner for security issues, because beside the handshake unique key the client must transmit at the beginning of the communication, multiple clients cannot connect if the socket is already in use. The used ports range from 8000 to 8007 because they usually represent unused ports on Debian running machines.

IV. Robot Design and Discussions

Fig. 6.a above illustrates the final assembly of the robotic arm and all the peripherals. Highlighted with red in the photo is the plexiglass casing which houses the electronic circuits. It can be seen in detail in Fig. 6.b.



(**a**)

(**b**)

Fig. 6. Designed and implemented robot: (a) 6DOF robotic arm; (b) Electronic circuits after assembly.

After all the circuits were construed and assembled, the connections made, screws tightened, cables connected, and code uploaded to the microcontrollers the system was up and running. When operating the switch present on the power supply, supposing the supply is plugged in, the power supply starts distributing power the entire system and all the afferent devices connected to the supply. When everything was said and done the system was working as intended, when receiving the appropriate command, the master acted accordingly. When an external command is transmitted to the Raspberry Pi, for example the command specifies to move the arm 400 to the left and open the gripper while at it, the master sends the command further to the slaves and using slave number 3 tasked with the control of the stepper, it moves the base 400 to the left, meanwhile the master sends the request to the first two slaves to control the stepper motors, ultimately opening the end effector of the arm. All the data from the sensors, current sensors and reflectance, is collected and transmitted in real time to the master so it can be later used for offline analysis. The results obtained after the completion are satisfactory, taking into consideration the limited budget it was built on and the fact that all the system was built in house by hand. Compared to other solutions, which tackle the same application [11], the results are somewhat similar to what other achieved with more expensive hardware and more time available [10].

V. Conclusions

After concluding the implementation and testing of the project, we have obtained several results regarding proper implementation of a robotic structure. The 3D printed materials offer acceptable flexibility and durability in complex structures, but as obtained from the current mechanical design, one can easily notice the elasticity of the material does decrease the overall system robustness resulting in unwanted oscillations and inertia produced movements. Nonetheless, this can be improved by implementing a more robust mechanical structure and designing one piece composed joints.

Another mechanical drawback was derived from the low quality of the chosen actuators. The low-cost servomotor batches ordered described a high nonlinear behavior, making them difficult to control and to obtain the desired outputs. Given the unreliability of the servomotors, the torque provided was lower than expected, resulting in the impossibility to control the heavier loaded joints of the robot, such as the lower base ones.

Mentioned in the II.C section were the concept of state machines and the design of the state machine, while here the implementation will be discussed. You might have noticed that in the previous chapter monitors were mentioned, but only in the testing phase; this is because they are slowing the program drastically and cannot be used in the final version of the code. A vital rule of working with finite state machines is avoiding using delays at all cost; the monitors and plotters included in the IDE also cause delays. The code is written using interrupt routines to avoid the use of delays, they cannot be present in a code that is made for a real-time application. As a bonus for the efficiency of the program, the code in the interrupt routines was kept as compact as possible to make the interrupt as fast as possible; it is worth mentioning we are talking about software interrupts, not hardware interrupts.

Acknowledgment

The research presented in this paper was supported by the following projects: ROBIN (PN-III-P1-1.2-PCCDI-2017-0734) and SeMed (2933/55/GNaC 2018).

References

[1] L. A. Grieco, A. Rizzo, S. Colucci, S. Sicari, G. Piro, D. Paola, and G. Boggia, "IoT-aided robotics applications: Technological implications target domains and open issues," *Computer Communications*, 2014, vol. 54, pp. 32-47, 2014

[2] R. A. El-laithy, J. Huang, and M. Yeh, "Study on the use of microsoft kinect for robotics applications," in *Proc. 2012 IEEE/ION Position, Location and Navigation Symposium*, pp. 23-26 April 2012

[3] R. Seiger, C. Seidl, U. Aßmann, and T. Schlegel, "A capability-based framework for programming small domestic service robots," in *Proc. the 2015 Joint MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-based Software-Engineering*, L'Aquila, Italy, pp. 49-54, 2015.

[4] What is 3D printing. [Online]. Available: https://ultimaker.com/en/explore/what-is-3d-printing

[5] E. Macdonald, R. Salas, D. Espalin, M. Perez, E. Aguilera, D. Muse, and R. B. Wicker, "3D Printing for the rapid prototyping of structural electronics," *IEEE Access*, vol. 2, pp. 234–242, 2014.

[6] ABS-Poly (Acrylonitrile, butadiene, styrene) property data. [Online]. Available: http://www.matweb.com/reference/abspolymer.aspx

[7] C. Chang, C. Liang, Z. Xuefeng, and Y. Wu, "Controlling a robot using leap motion," in *Proc. 2nd International Conference on Robotics and Automation Engineer*ing, pp. 48-51, 2017.

[8] Epson Exceed Your Vision, Epson LS3 SCARA Robots - 400mm. [Online]. Available: https://epson.com/For-Work/Robots/SCARA/Epson-LS3-SCARA-Robots---400mm/p/RLS3401ST9P5

[9] F. Wagner, R. Schmuki, T. Wagner, and P. Wolstenholme, *Modeling Software with Finite State Machines*, Publisher: Auerbach Publications Boston, MA, USA, 2006, ISBN:0849380863

[10] C. T. Kiang, A. Spowage, and C. K. Yoong, "Review of control and sensor system of flexible manipulator," *Journal of Intelligent & Robotic Systems*, 2014, pp. 187–213

[11] J. Lee, P. H. Chang, and M. Jin, "Adaptive integral sliding mode control with time-delay estimation for robot manipulators," *IEEE Transactions on Industrial Electronics*, pp. 6796 – 6804, 2017.

Stan O. is a lecturer in the Automation Department at the Technical University of Cluj-Napoca. His research interests include medical informatics, semantic interoperability, information management in the age of the Internet, dependability and fault-tolerant systems.

Stan received a PhD in systems engineering from the Technical University of Cluj-Napoca. He is a member of IEEE.

Manea A. is a student in the Automation Department at the Technical University of Cluj-Napoca. His research interests include design embedded systems and embedded software.