Fuzzy system as a method of controlling LEGO Linefollower vehicle using C# programming language

Krzysztof Grzesica[a], Jakub Wadas[a]

[a] Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, 44-100 Gliwice

Abstract

In this article we present our implementation of intelligent system developed for a LEGO robot. We have built a robot which is using sensors to trace the line and follow it. The model of tracing is based on fuzzy rules, which have been done to follow the shape of a black line on the white background. The system was implemented using C# language for LEGO Mindstorm elements. Results of our experiments show that the robot is able to follow various shapes of black lines.

Keywords

LEGO robot, Line detection, Fuzzy rules

1. Introduction

In many fields of industry and production various robotics are applied to help in situations where the work may be dangerous for humans or there is required high precision at work. To start the research in such field we have selected LEGO mindstorm elements. This robot model is easy to develop just by using simple bricks in many shapes from the set. The set also provides a programmable electronics with a variety of sensors. On the other hand such approach is widely presented in other literature, what gave us an inspiration to develop our idea.

In [1] a model of LEGO blocks was used to work with human gestures, where sensors were used to read gestures and therefore take actions in mindstorm elements. In [2] was presented how to use such LEGO models to help in first contact with machine intelligence. Authors describe good practices when using LEGO Mindstorm as a platform for programming artificial intelligence systems. As presented in [3] these models also support creative thinking, especially when we develop new algorithms that must be programmed in a special way accepted by the LEGO Mindstorm platform. There are various approaches to use programming languages in developing such robots.

However the most necessary is tu use an optimal library, which will support all necessary function that provide optimal connection and configuration between elements of the robot. In [4] was discussed how to select such libraries and optimize them for robotic constructions. Also some approaches are based on frameworks, as proposed in [5] that frameworks are also possible for other constructions based on arduino elements, which provide similar possibilities of robotic constructions. All these what we want to create depends on our ability to develop the construction and than to implement the control software. An interesting discussion was given by [6, 7].

The other aspect of robotic models is to select a proper decision support model. In many IoT constructions we can find various approaches. In [8] was proposed how to combine neural networks with rules in a form of soft set table. While in [9] the fuzzy rule system was implemented with neural networks to work in a smart house environment. It is also very popular to develop fuzzy rules working on images, as discussed in [10]. Mechanical vehicle constructions also very often benefit from artificial intelligence, both at construction and simulation level, as proposed in [11, 12, 13, 14].

Our approach is using C# language to implement control system based on fuzzy rules to first detect and than follow the line. The LEGO Mindstorm robot is using two servo-motors as accelerators of wheels controlled by our developed artificial intelligence to follow the line. our experiment shows that our construction is well defined and can follow even the complex lines on the board.

2. Model

In this section we will describe how we have developed the model both from programming side and thinking model.

SYSTEM 2020: Symposium for Young Scientists in Technology, Engineering and Mathematics, Online, May 20 2020

krzygrz684@student.polsl.pl (K. Grzesica);
jakuwad985@student.polsl.pl (J. Wadas)

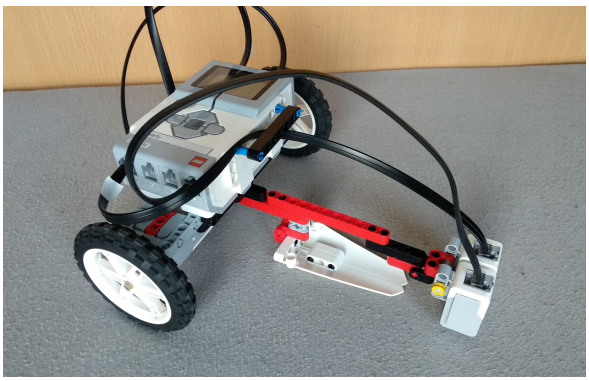

Figure 1: Vehicle construction

2.1. Project assumptions

The aim of this project was to build an artificial intelligence system based on fuzzy logic, which would steer the vehicle used in LEGO Line-follower competition. Based on readings, the system must decide how to change the speed of each engine, so that:

- The vehicle does not go off the road (A moment when a black line is not between vehicle's wheels is considered going off the road)
- The vehicle goes as fast as possible

The robot is connected to computer via Bluetooth. The robot sends raw readings and data to computer where all necessary calculations are being performed. Then, based on calculation results appropriate commands from the computer are sent to robot.

2.2. Robot Construction

To accomplish that task we have built a vehicle using LEGO parts. The robot is based on intelligent EV3 brick, it is equipped with two color sensors, which measure the level of reflected light they emit. The vehicle is driven by two independent engines which directly spin the wheels. The actual look of vehicle can be seen in Fig. 1 and Fig. 2.

3. Mathematical Model

3.1. Normalization

Firstly, we must normalize the readings from color sensors to negate the hardware differences. To do that we must first calculate parameters P_L and P_R accordingly as:

$$P_L = \frac{100}{WCL - BCL} \tag{1}$$

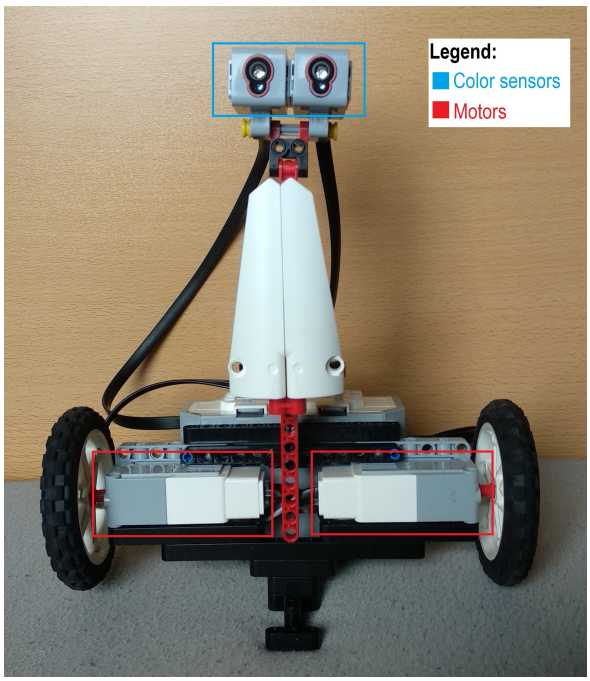

Figure 2: The bottom of the vehicle

$$P_R = \frac{100}{WCR - BCR} \tag{2}$$

Where WCL and BCL are accordingly, maximum and minimum reading value of left sensor. WCR and BCR are maximum and minimum reading values of right sensor. These values are obtained during setup configuration. Reading output is determined using the following formula:

$$readL = |(readingLeft - BCL) * P_L - 100| \tag{3}$$

$$readR = |(readingRight - BCR) * P_R - 100| \tag{4}$$

Where $readingLeft$ and $readingRight$ are respectively raw vuales returned by left and right color sensor. $readL$ and $readR$ take values from 0 to 100, where 0 means total white, 100 - pure black.

After that, the resultant reading value meaning the inclination of the road is calculated accordingly to the following formula:

$$reading = \frac{readR - readL}{10} \tag{5}$$

If both color sensors went off the road the variable $reading$ i set to 10 or -10 depending on the side of road the sensors are located.

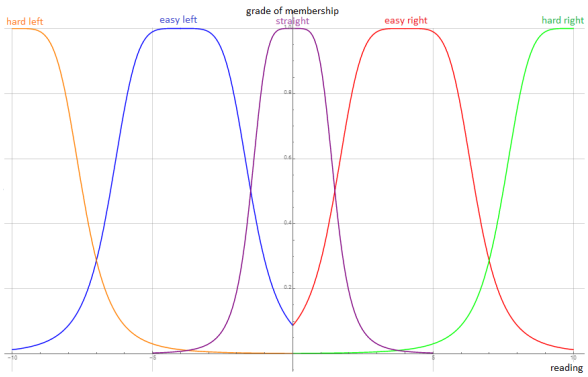

Figure 3: Reading membership functions

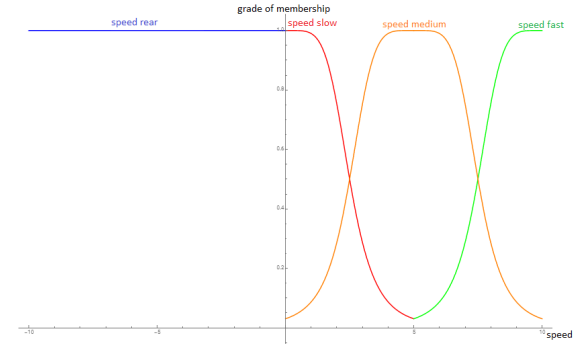

Figure 4: Speed membership functions

3.2. Fuzzyfication

To determine grade of membership Bell shaped and Linear functions are used. The formulas are as follows:

$$Bell = \begin{cases} \frac{1}{1+|\frac{x-c}{a}|^{2b}} & , y \leqslant x \leqslant z \\ 0 & , x < y \vee x > z \end{cases} \tag{6}$$

$$Linear = \begin{cases} 1 & , x < 0 \\ 0 & , x \geqslant 0 \end{cases} \tag{7}$$

Where a, b are parameters. Variables y and z are endpoints of interval. Variable c stands for center of function. In our project $b = 2.5$ and $a = 2.5$ in all but one membership function. The membership function Straight uses $a = 1.5$. The reading membership functions can be seen in Fig.3, while speed membership functions in Fig. 4.

Then, the resultant vehicle speed composed from separate engines speeds is being calculated. In order to determine it we calculate each engine's grade of membership for all of speed membership functions and accordingly to fuzzy rules base shown in Tab. 1 we conduct necessary calculations.

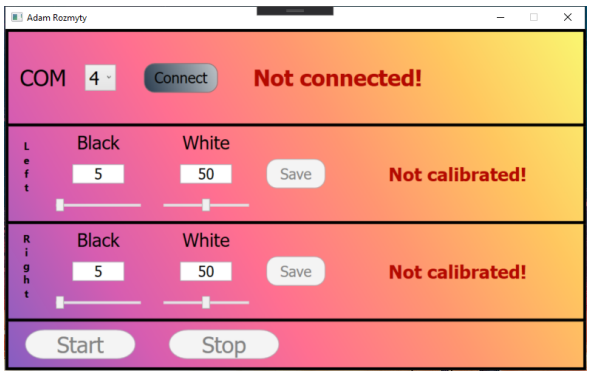

Figure 5: Application window before configuration

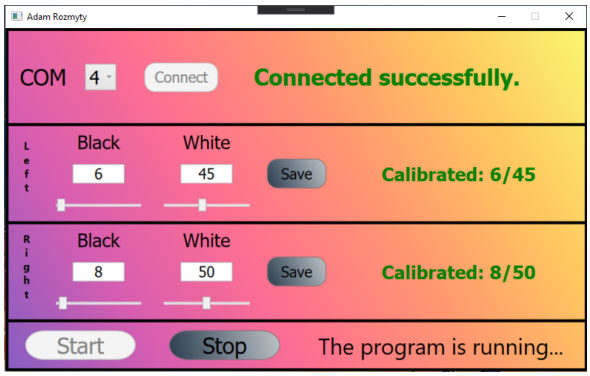

Figure 6: Application window after configuration

Table 1
Speed base of rules

Left/Right	rear	slow	medium	fast
rear	rear	slow	slow	slow
slow	slow	slow	medium	medium
medium	slow	medium	medium	fast
fast	slow	medium	fast	fast

The results are being grouped by the rules they correspond with and for each of rule we choose the biggest value. Lets assume that the biggest value for rule slow is u_w, for medium - u_m and for fast - u_f. The final resultant speed is obtained by using center of gravity method defined as:

$$speed = \frac{u_w * p_s + u_m * p_m + u_f * p_f}{u_w + u_m + u_f} \tag{8}$$

where $p_s = 10$, $p_m = 50$ and $p_f = 90$ are coefficients of speed rules.

Table 2
Inclination base of rules

Speed/Reading	HL	EL	S	ER	HR
fast	HL	HL	S	HR	HR
medium	HL	EL	S	ER	HR
slow	HL	EL	S	ER	HR

HL - Hard Left, EL - Easy Left, S - Straight,
ER - Easy Right, HR - Hard Right

```
static public double Reading(int readingLeft, int readingRight)
{
    double readL = Math.Abs(((readingLeft - BlackColorLeft) * P_L) - 100);
    double readR = Math.Abs(((readingRight - BlackColorRight) * P_R) - 100);

    if (readingLeft > WhiteColorLeft/2 && readingRight > WhiteColorRight/2)
    {
        if (previousTurn)
            return -10;
        else
            return 10;
    }
    if (readL > readR)
        previousTurn = false;
    else
        previousTurn = true;

    return ((readR - readL) / 10);
}
```

Figure 7: Reading method

3.3. Deffuzyfication

The next step is to calculate the inclination, which consists of resultant speed and resultant reading values. In order to determine it we calculate resultant speed grade of membership for all of speed membership functions and resultant reading grade of membership for all of reading membership functions. Then, accordingly to fuzzy rules base shown in Tab. 2 we conduct necessary calculations. The results are being grouped by the rules they correspond with and for each of rule we choose the biggest value. Lets assume that the biggest value for rule HL is u_1, for EL - u_2, for S - u_3, for ER - u_4 and for HR - u_5. The final inclination is obtained by using center of gravity method defined as:

$$inc = \frac{u_1 * p_1 + u_2 * p_2 + u_3 * p_3 + u_4 * p_4 + u_5 * p_5}{u_1 + u_2 + u_3 + u_4 + u_5} \tag{9}$$

where $p_1 = -100$, $p_2 = -50$, $p_3 = 0$, $p_4 = 50$, $p_5 = 100$ are coefficients of inclination rules accordingly for HL, EL, S, ER and HR.

Finally, the rule with the greatest inclination grade of membership value is chosen. Depending on it, the engines speeds are calculated and set. For Hard Left (10) and (11), for Easy Left (12) and (13), for Straight (14) and (15), for Easy Right (16) and (17) and for Hard Right (18) and (19).

$$LeftMotor = \frac{inc}{4} \tag{10}$$

$$RightMotor = inc \tag{11}$$

$$LeftMotor = inc * 1.5 \tag{12}$$

$$RightMotor = inc * 2 \tag{13}$$

$$LeftMotor = 100 \tag{14}$$

$$RightMotor = 100 \tag{15}$$

$$LeftMotor = inc * 2 \tag{16}$$

$$RightMotor = inc * 1.5 \tag{17}$$

$$LeftMotor = inc \tag{18}$$

$$RightMotor = \frac{inc}{4} \tag{19}$$

where inc is the inclination value calculated before. In case of Easy Left and Easy Right if the inc value is lesser than -50 or greater than 50, then it is set to 50.

4. Implementation

Let us now present the software we have done.

4.1. Application

Based on the mathematical model described above, we have created a robot control application. The application has a very simple graphical user interface, which makes it user-friendly. In Fig. 5 and Fig. 6 application window can be seen.

4.2. Program code

The program is written in *C#*. In addition to the standard *.NET* libraries, the program uses the *Lego.Ev3* library for communication between the computer and the robot. The program has been divided into classes and appropriate methods.

Reading method This method is responsible for the normalization of color sensor data. This method also serve as a precaution against loosing the route by a robot. Method code can be seen in Fig. 7.

HowTheRouteRuns method This method is a fragment of the fuzzy system. It combines the route inclination with the current vehicle speed and decides how to react based on that data. A piece of the method code is in Fig. 8.

Robot class This is the class which object represents vehicle instances in the program. The most important class fields and properties representing the state

```
public static double HowTheRouteRuns(double speed, double reading)
{
    const int hardLeft = -100;
    const int easyLeft = -50;
    const int straight = 0;
    const int easyRight = 50;
    const int hardRight = 100;

    speed /= 10;
    #region HardLeft
    double hardLeftMembership;
    double temp;
    temp = ElementaryFunctions.ReadingHardLeft(reading) *
        ElementaryFunctions.SpeedFast(speed);
    hardLeftMembership = temp;
    temp = ElementaryFunctions.ReadingHardLeft(reading) *
        ElementaryFunctions.SpeedMedium(speed);
    if (temp > hardLeftMembership)
        hardLeftMembership = temp;
    temp = ElementaryFunctions.ReadingEasyLeft(reading) *
        ElementaryFunctions.SpeedFast(speed);
    if (temp > hardLeftMembership)
        hardLeftMembership = temp;
    temp = ElementaryFunctions.ReadingHardLeft(reading) *
        ElementaryFunctions.SpeedSlow(speed);
    if (temp > hardLeftMembership)
        hardLeftMembership = temp;
    #endregion
    #region EasyLeft
    #region straighttraight
    #region HardRight
    #region EasyRight
    double value = (easyLeftMembership * easyLeft + easyRightMembership *
        easyRight + hardRightMembership * hardRight + hardLeftMembership *
        hardLeft + straightMembership * straight) / (easyLeftMembership +
        easyRightMembership + hardRightMembership + hardLeftMembership + straightMembership);
    return value;
}
```

Figure 8: HowTheRouteRuns method

```
public class Robot
{
    private Brick _brick;
    public bool Connected { get; private set; }
    public event Action ConnectionSuccess;
    private bool isDriving;
    private uint leftSensor;
    private uint rightSensor;
    private int _leftMotor = 40;
    private int _rightMotor = 40;
(...)
```

Figure 9: Robot class

```
public async Task Go()
{
    isDriving = true;

    while (isDriving)
    {
        await Task.Delay(70);
        _brick.BatchCommand.TurnMotorAtPowerForTime(OutputPort.A, RightMotor, 30, false);
        _brick.BatchCommand.TurnMotorAtPowerForTime(OutputPort.D, LeftMotor, 30, false);
        await _brick.BatchCommand.SendCommandAsync();
        double robotSpeed = Speed.GetResultantSpeed(LeftMotor, RightMotor);
        double reading = ElementaryFunctions.Reading((int)leftSensor, (int)rightSensor);
        double inclination = Core.HowTheRouteRuns(robotSpeed, reading);
        ChooseTurn(inclination);
    }
}
```

Figure 10: Go method

of the object can be seen in Fig. 9. *Brick* class object, which belongs to the *Lego.Ev3* library represents the LEGO EV3 brick. *Connected* property represents the state of connection between the program and the robot. Fields *leftSensor* and *rightSensor* store values obtained from color sensors. Fields *_leftMotor* and *_rightMotor* store current motors speeds.

Go method This method belongs to the *Robot* class. The method is responsible for the robot's movement. It contains loop in which the fuzzy system makes calculations, decides about the next move and finally makes that move by setting the power of motors. The full code of the method can be found in Fig. 10.

ChooseTurn method This method belongs to the *Robot* class. It is called in the loop described above.

```
private void ChooseTurn(double inclination)
{
    List<Action<double>> decisions = new List<Action<double>>();
    List<double> memberships = new List<double>();

    memberships.Add(ElementaryFunctions.ReadingHardLeft(inclination/10));
    decisions.Add(TurnHard);
    memberships.Add(ElementaryFunctions.ReadingEasyLeft(inclination/10));
    decisions.Add(TurnEasy);
    memberships.Add(ElementaryFunctions.ReadingStraight(inclination/10));
    decisions.Add(GoStraight);
    memberships.Add(ElementaryFunctions.ReadingEasyRight(inclination/10));
    decisions.Add(TurnEasy);
    memberships.Add(ElementaryFunctions.ReadingHardRight(inclination/10));
    decisions.Add(TurnHard);

    int index = memberships.IndexOf(memberships.Max());
    decisions[index](inclination);
}
```

Figure 11: ChooseTurn method

```
private void TurnHard(double inclination)
{
    _brick.DirectCommand.PlayToneAsync(100, 200, 50);
    if (inclination < 0)//Left
    {
        inclination = Math.Abs(inclination);
        LeftMotor = (int)inclination / 4;
        RightMotor = (int)inclination;
    }
    else//Right
    {
        RightMotor = (int)inclination / 4;
        LeftMotor = (int)inclination;
    }
}
```

Figure 12: TurnHard method

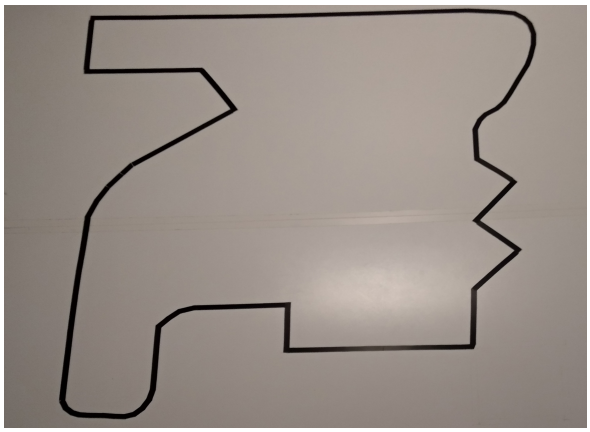

Figure 13: Test route

The method is responsible for choosing the direction in which the robot should go. The code for this method is shown in Fig. 11.

TurnHard method This method also belongs to the *Robot* class. It is responsible for updating the variables representing the current motors power. The full code of the method can be seen in Fig. 12. The *Robot* class also has *TurnEasy* and *GoStraight* methods whose operations are analogous to the *TurnHard* method.

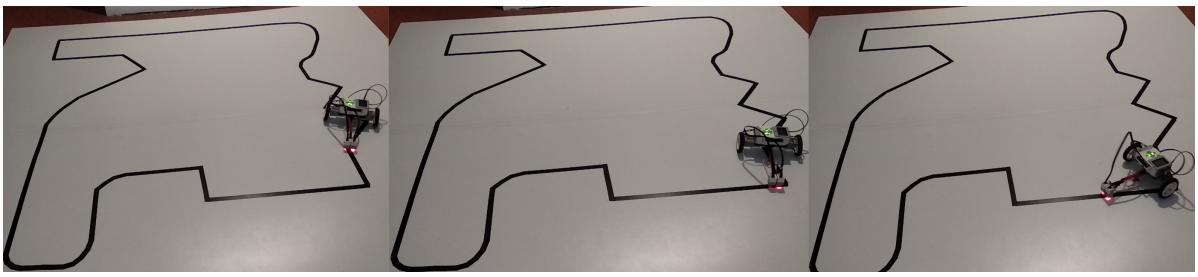

Figure 14: A part of a test ride

5. Tests

In order to test how the vehicle performs we created a test route (Fig. 13). The route is a 19-millimeter-wide black line on a white background. It is quite complicated due to a lot of 90° angle turns, it does not contain crossroads though. The vehicle runs through the route precisely but with little speed. It is worth mentioning that the system is incredibly sensitive to any changes in lightning and the non-uniformity of it. A part of a test ride can be seen in Fig. 14.

6. Conclusion

Because of poor functioning of some of *Lego.EV*3 library functions that our project was based on, the results were not as good as we had imagined. Commands must have been initialized for a specified time amount which led to delays and unstable movement of vehicle. What is more, in order to work properly the program had to use function *Delay*(), which stops the robot suddenly for few milliseconds.

Considering that *Lego.EV*3 library has not been updated for 7 years and it's author officially abandoned the project, rewriting the whole project to *RobotC* language seems to be the best way to improve program's performance, allowing the program to run on EV3 brick itself. Apart from solving problems mentioned above, that approach would also terminate problems connected with Bluetooth connection latency.

References

[1] L. I. Kovács, Gesture-driven lego robots, Acta Universitatis Sapientiae, Informatica 11 (2019) 80–94.

[2] Á. Martínez-Tenor, A. Cruz-Martín, J.-A. Fernández-Madrigal, Teaching machine learning in robotics interactively: the case of reinforcement learning with lego® mindstorms, Interactive Learning Environments 27 (2019) 293–306.

[3] N. L. Fanchamps, L. Slangen, P. Hennissen, M. Specht, The influence of sra programming on algorithmic thinking and self-efficacy using lego robotics in two types of instruction, International Journal of Technology and Design Education (2019) 1–20.

[4] A. Spanò, A. Cortesi, Legodroid: A type-driven library for android and lego mindstorms interoperability, Sensors 20 (2020) 1926.

[5] J. Vega, J. M. Cañas, Pybokids: An innovative python-based educational framework using real and simulated arduino robots, Electronics 8 (2019) 899.

[6] J. Wang, X. Du, H. Wang, Research & implementation of multitasking lego robots, in: 2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM), IEEE, 2019, pp. 655–659.

[7] R. Giuliano, G. Cardarilli, C. Cesarini, L. Di Nunzio, F. Fallucchi, R. Fazzolari, F. Mazzenga, M. Re, A. Vizzarri, Indoor localization system based on bluetooth low energy for museum applications, Electronics (Switzerland) 9 (2020) 1–20.

[8] M. Woźniak, D. Połap, Soft trees with neural components as image-processing technique for archeological excavations, Personal and Ubiquitous Computing (2020) 1–13.

[9] M. Woźniak, D. Połap, Intelligent home systems for ubiquitous user support by using neural networks and rule based approach, IEEE Transactions on Industrial Informatics (2019).

[10] G. Capizzi, G. Lo Sciuto, C. Napoli, D. Polap, M. Woźniak, Small lung nodules detection based on fuzzy-logic and probabilistic neural network with bio-inspired reinforcement learning, IEEE Transactions on Fuzzy Systems 6 (2020).

[11] J. T. Starczewski, P. Goetzen, C. Napoli, Triangular fuzzy-rough set based fuzzification of fuzzy

rule-based systems, Journal of Artificial Intelligence and Soft Computing Research 10 (2020) 271–285.

[12] M. Woźniak, D. Połap, Hybrid neuro-heuristic methodology for simulation and control of dynamic systems over time interval, Neural Networks 93 (2017) 45–56.

[13] G. Capizzi, F. Bonanno, C. Napoli, Hybrid neural networks architectures for soc and voltage prediction of new generation batteries storage, 2011, pp. 341–344.

[14] F. Bonanno, G. Capizzi, C. Napoli, Some remarks on the application of rnn and prnn for the charge-discharge simulation of advanced lithium-ions battery energy storage, 2012, pp. 941–945.