# Open Source Interfacing for LabVolt 5250 Robotic Arm

Mohammed Elsharif, Nabil Drawil, and Mahmoud Osman
Department of Computer Engineering, Faculty of Engineering
University of Tripoli, Tripoli, Libya
mohammed.elsharif@gmail.com, n.drawil@uot.edu.ly

***Abstract* – Open source robotics has been attracting great attention recently. Indeed, Not only students and researchers, but also manufacturing sectors has been among the biggest beneficiaries of the robot renaissance supported by open sourcing.**

**Sparki, ArduPilot Copter, and TurtleBot projects are examples of avialable open source hardware platforms. However, none of them approximate what industrial robots really are, like LabVolt 5250 Robotic Arm does – An affordable training system that cover the programming and operation of industrial robots.**

**LabVolt 5250 is known for being suitable for learning industrial automation with very limited capabilities for conducting advanced research in fields such as artificial intelligence and optimization. This urges the need for the development of open source interfacing with LabVolt 5250.**

**In this paper, the low level of communication protocol with the Robotic Arm Control Unit (RACU) is presented. An open source Matlab library is also developed to further provide the utility and tools needed by future developers.**

*Index Terms*—**Terms—LabVolt Robotic Arm, Serial Port Communication, Articulate System, Matlab.**

## I. INTRODUCTION

Recent development in automation and Robotics is paving the way towards a more vibrant future, where results are currently observed in reallife. However, the amount of growth and development can be limited without providing an open environment [1]. LabVolt 5250 Robot Arm is a prime example of a system that is designed for simple educational asks, which can be developed and automated to be used in advanced applications. This research work focuses on open sourcing and interfacing the LabVolt 5250 Robotic. such achievement is a hurdle removal for many advanced robotic systems, which cognitively deals with objects in its surrounding. Capabilities of perception and approaching dynamically located objects, based on specific attributes, are just a few potential features of future robots. With the Provided tools in this research, future developers can focus on more abstract issues related to higher level of artificial intelligence. Whereas, the whole concern of dealing with the lower layers of serial communication, data control or exception handling has been taken in consideration herein.

Two main contributions provided in this research work. The first contribution is the provision of the low-level communication protocol between a computer and the control unit of the Robotic Arm. This knowledge is attained after extensive investigation on the key commands of the robotic arm control unit. Second contribution is the development of open source high-level Matlab library that contains functions to master the controller of the Robotic Arm.

The remainder of this paper is organized as follows. This section focuses on understanding the LabVolt Robotic Arm system and examining its different features and modes. Section II presents an analyzes of the communication protocol between the official LabVolt software, on a personal computer, and the Robot Arm Control Unit (RACU). The objective of the protocol is to control the Robotic Arm movements. Section III introduces the developed Robotic Arm Library using Matlab, which facilitate the communication with RACU and control the Robotic Arm. This Library is aimed to provide developers and students the needed primitives, and to take care of low level control, exception handling while communicating with RACU. Finally, Section IV provides concluding remarks.

The LabVolt 5250 Robotic Arm is comprised of three main modules which are described in Fig. 1.

### *A. LabVolt Servo 5250 Robotic Arm*

An articulated-arm robot that has five axes of rotation and six servo drives with closed-loop control via optical encoders, Fig. 1-(a). The robotic Arm uses several joints simultaneously to perform a programmed move sequence controlled by the RACU. Movement is accomplished with belts through a series of gears and belt-driven pulleys.

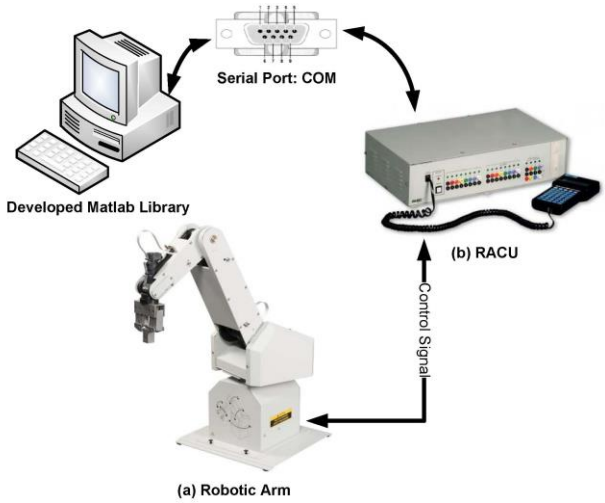


Figure 1 System Modules

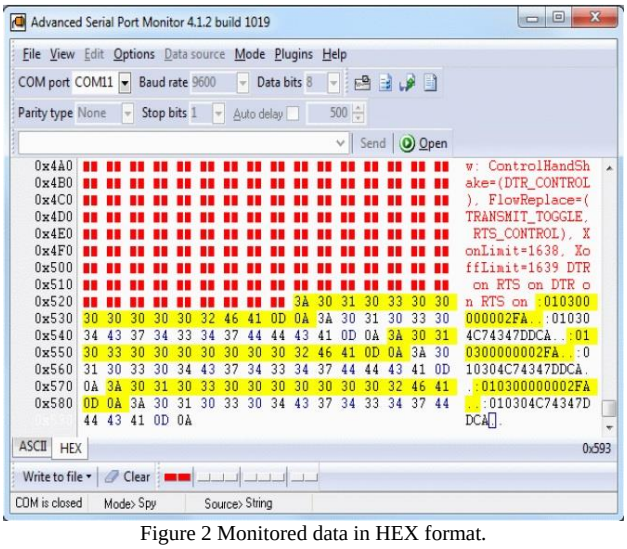


Figure 2 Monitored data in HEX format.

The 5250 can operate within a maximum radius of 732 mm (28.5 in). It has a 1.4-kg (3-lb) load capacity and weighs 20.4 kg (45 lb) [2].

### *B. Robotic Arm Controller Unit (RACU)*

The RACU unit is an embedded microprocessor with connections for inputs and outputs as well as a floppy disk drive for saving task/point programs, which is shown in Fig. 1-(b). The RACU controls the Robotic Arm servo motors through a closed loop PID controller [2]. RACU is operated from a handheld terminal, or the robotic software, such as RoboCIM.

### *C. RoboCIM Software*

RoboCIM is a software used to simulate and control the operation of the Servo Robot System Model 5250. The software is also used to design Robot Arm movement routines for different tasks. In RoboCIM, the actual equipment is replaced with three dimensional-images. Sophisticated mathematical models accurately simulate the mechanical and electrical characteristics of the equipment. One of the RoboCIM characteristics is to control and visualize the motion of the system interactively [2]. RoboCIM Software exchanges information with RACU using RS-232 standard communication protocol. RS-232 is the defacto standard solution for linking computer and its peripheral devices to allow serial data exchange. It specifies common voltage and signal level, common pin wire configuration and minimum amount of control signals [3].

## II. INTERFACING THE ROBOT

This section analyzes the communication between RoboCIM and the RACU by monitoring the active communication process, decoding the monitored data and discussing the found results and features.

### *A. Port Monitoring*

Determining the type of connection and understanding the communication protocol between RoboCIM and the RACU is the first step for successful interfacing.

The RACU is connected with the computer through a Serial Communication Port COM, using RS232 protocol. RoboCIM uses COM to communicate with the RACU, which in turn controls the servo motors in the Robotic Arm, with accurate feedback.

The data transmission through COM between RoboCIM and the RACU must be monitored for further analysis. But this task can not be accomplished without knowing a number of parameters that include baud rate of transmission between COM and RACU, data bit packet characteristics, end bit numbers, and type of flow control. Once determined, data can be successfully monitored and exported to a file for decoding and analysis. The assumption is that a movement function, for example, always results in the same code combination, once called again by RoboCIM.

Advanced Serial Monitor is the software used for serial port monitoring. Advanced Serial Port Monitor allows sniffing and monitoring COM ports of various types, including RS-232, RS485 and RS422 serial interfaces. Advanced Serial Port Monitor provides many powerful functions and features that support an intricate serial port monitoring tasks.

Signals that control the RACU can be determined by monitoring the communication between the RACU and the COM while the RoboCIM is running and active. When RoboCIM sends a movement signal for example, Advanced Serial Monitor monitors the transmitted code and exports it to a text file. The Monitored data at first glance, can look like a scrambled set of code as shown in Fig. 2, This raises the question of encryption, as a method to obscure communication.

Monitored code can be tested and decoded by recognizing the commonalities and recursions between the code while examining and repeating different commands using RoboCIM.

### *B. Findings of Port Monitoring*

After testing, it was determined that the RS-232 mode of communication between RoboCIM and RACU is in asynchronous mode with no flow control. The COM parameters are selected after try and error attempts till

TABLE 1 CONFIGURATION OF COM PORT.

| | |
|---|---|
| Baud rate | 9600 |
| Data Packet Size bit | 8 Bits, 1 End |
| Flow control | none |
| End Byte | 0D |

Note that the last ASCII character in the aforementioned command is "0D". It is understood that "0D" signifies end of command sent by RoboCIM. On the other hand, the end of string is "0A" when serial data received from RACU instead. Such conclusions are obtained after many observations of different arm movement commands.

In addition, the carriage return character and its equivalent ASCII code give the clue that RACU might be dealing with ASCII codes. Therefore, the monitored HEX data is translated into ASCII characters and commands. When the code was changed to its ASCII equivalent, the monitored code immediately becomes understandable, and easier to decode, with assembly like programming commands. For example, "4d 4f 56 45 50 05 01 0D ", becomes "MOVE 50 05 01 0D", which is easier to decode and understand, as shown in Table 2. The same process is repeated for different functions and modes in RoboCIM and all the commands were successfully found and noted.

Table 2 MOVEMENT COMMAND DISSECTION.

| HEX Code | Function |
|---|---|
| 4d 4f 56 45 | Move Command |
| 50 Command Separator | Command Separator |
| 05 | Servo Motor Identifier |
| 01 | Direction of Movement |
| 0D | Carriage Return |

After successfully decoding the data, and noting all the commands. The noted commands, are tested, by communicating with the RACU directly through the COM port session using Advanced Serial Port Monitor alone. The Robotic Arm was successfully controlled by sending the found commands without RoboCIM. However, for successful control the control commands must be sent precisely as found in the code. If a part was missed or added, buffer overflow might occur and the RACU will crash and require manual reboot.

### C. Robotic Arm Control Commands

The Robotic Arm can be interfaced using any programming language if the following rules of communication are followed:

- Commands must be sent exactly as found.
- Commands must end with a line terminator "0D".
- Commands cannot be received when RACU is executing.
- Only STOP command can interrupt RACU when executing.

Consequently, RACU interpreter will remain waiting if "0D" was not sent. Also, RACU responds by "BSY" or "ERR" message if a command was sent before the completion of the previous one. STOP command interrupts any current execution and immediately stops the Robotic Arm. In Table 3, shows the list the commands used to operate RACU directly through the COM port.

### D. RACU Coordinate Transform

The RACU uses a special articulate coordinate system, that only uses integers, $n_{joint}$ , to define robot join angles. For example, it is found that number 64995 in RACU articulate coordinates signifies 89.9 degrees in conventional articulate coordinates. Hence, there must be a linear transformation from RACU articulate coordinates to world articulate coordinates and vice versa. Each joint has its own linear transformation. Equations (1) to (5) are derived by finding the corresponding mapping between the integer based angle, $n_{joint}$ , and the degree based angle, $\theta_{joint}$ , of each joint.

$$\theta_{wrist} = -n_{wrist} \times 0.001699 \quad (1)$$

$$\theta_{elbow} = -\frac{n_{elbow}}{666.66667} \quad (2)$$

$$\theta_{shldr} = 105 + n_{shldr} \times 0.0015 \quad (3)$$

$$\theta_{base} = \frac{n_{base}}{666.66667} \quad (4)$$

$$\theta_{gripper_{rotate}} = -\frac{n_{egripper_{rotate}}}{525} \quad (5)$$

## III. MATLAB ROBOTIC ARM LIBRARY

This section introduces Robotic Arm Library, written in Matlab. This open source library is made available in [4]. The aim of this library is to facilitate communication with the RACU and control the Robotic Arm. Using the decoded commands from Section II a library is developed so that it can be utilized independently from any other software such as RoboCIM. Matlab is chosen for this task as it is a high-performance language for technical computing. Matlab integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.

Furthermore, Matlab Instrument Toolbox includes, a number of low level serial port manipulating functions, that can be used to send signals through the COM port, with the needed specification. These low-level I/O functions are the backbone for the Robotic Arm Control Library.

The developed functions are categorized and described in the following subsections.

### A. Robotic Arm Initialization and Shutdown

The first step of using the Robotic Arm is to initialize the Serial Port connection between Matlab and the RACU. This step will create a serial object and adjust the connection parameters to be compatible with the RACU. The parameters of interest are listed in Table 1.

Table 3 BASIC COMMANDS TO OPERATE RACU.

| Command | Interpretation | Input | Output |
|---|---|---|---|
| `remote 0D` | Check for status of the robotic arm in terms of being Ready or Disconnected | none | `OK` or `ERR` |
| `hardhome 0D` | Put the RACU and robotic arm in sync by moving all joints to physical limits and then move to home position | none | Keep returning the corrected joint position till reaches the home position, closes by send `END` message |
| `shutdown 0D` | Shutdown the RACU and robotic arm | none | Closes with `END` message |
| `mov 50 Y X 0D` | Move one selected joint $Y$ in direction $X$ till a STOP order is initiated or reach physical limit of that joint | $Y \in \{0,1,2,3,4,5\}$ where 0 ≡Gripper Open/Close, 1 ≡ Gripper Rotation, 2 ≡Wrist, 3 ≡ Elbow, 4 ≡Shoulder, and 5 ≡ Base $X \in \{1,-1\}$ where 1 ≡ Clockwise and −1 ≡Unticlock wise | none |
| `stop 0D` | Override any order and immediately stop the robotic arm | none | none |
| `set estop 0 0D` | Release the emergency stop lock. This command is used to resume a session after the physical emergency button was pressed and physically released. | none | none |
| `free 0D` | Free servo motors if torque. | none | `OK` if successful, `ERR` if en error occurs |
| `torque 0D` | Torque servo motors if free | none | `OK` if successful, `ERR` if en error occurs |
| `Get POS 0D` | Return the current articulate position of the arm | none | `P A B C D E F 0 0` where `P` signifies position array. The coordinates are presented with articulate values and `0 0` as ending bytes A≡Base, B≡Shoulder, C≡Elbow, D≡Wrist, E≡ Gripper-R, F≡Gripper-L |
| `run 50 0 A B C D E F 0 0 1 0D` | Go to specified articulate position, in this command all servo motors rotate concurrently to reach the designated location. | 50 is a separator, A Gripper Open/Close, B≡ Gripper rotation, C≡ $n_{wrist}$, D≡ $n_{elbow}$, E≡ $n_{shoulder}$, F≡ $n_{base}$, `0 0 1 0D` ending characters | Robotic Arm returns its position when the final position is reached. |

Matlab deals with COM ports as an object that can be defined and adjusted based on the specific application required, the COM object must be opened after creation and closed before termination. The function Obj = Init_Robot("Port Name") must be called to start using the Robotic Arm Library. The function reserves and initializes the COM port with the RACU specified settings and open the specified port for use. The function returns an Object reference to COM port.

After finishing with the Robotic Arm, the serial port COM connected with the RACU must be shutdown by releasing the COM port object and initiating a shutdown signal. The RACU will release the connection from its side and shutdown. The following function is used to terminate Matlabs connection with the COM port and release it for final shutdown: `States = Close_Robot(Obj)`. The states of the Port is returned to the invoking function.

### *B. Robotic Arm Calibration*

After initialization, the RACU must be calibrated using the `Calibrate_Robot()` function developed in the Matlab Robotic Arm Library. Calibration is imperative as it puts the Robotic Arm and RACU in synchronous in terms of their respective coordinate spaces; otherwise, movement commands will produce erroneous action in the physical world.

The calibration process is conducted by resetting the joint counters, i.e., $n_{joint}$ , while rotating the physical joints to the physical limit of the respective servo motor. For example, when calibrating the base motor, the robotic arm will rotate towards the right limit until the right limit sensor of the base motor activates. Once reached, the RACU sets $n_{base}$ to the limit respective value. The Robotic arm, then, rotates towards the left limit until reaches the home position of base joint. The same process is repeated with all servo motors ending in the final calibrated position as shown in Fig. 3.

Figure 3 Robotic Arm calibrated position.

Function Pos = Calibrate_Robot(obj) is a blocking function, because Matlab is blocked when invoking this function until the end of the calibration processes. The reason behind blocking Matlab during calibration is to prevent any signal from being sent to the RACU. If a signal is sent while the arm is busy, the RACU will send a busy "BSY" signal or in some cases crashes. The function returns the location of the arm after calibration is done, which is "0, 0, 0, 105, -0.0015".

### *C. Robotic Arm Positioning*

The position targeted by the Robotic Arm can be represented by the two coordinate systems, namely, Articulate system and Coordinate system.

1) Articulate system: Articulate system defines objective location in terms of joint angles in space. Every angle represents an orientation that adds to the final end effector position: starting from the wrist angle and ends at base angle, as shown in Fig. 4. To conduct a movement to a target location in space, five $\theta$'s must be defined ( $\theta_{base}$ , $\theta_{shldr}$ , $\theta_{elbow}$ , $\theta_{wrist}$ , $\theta_{gripperrotate}$ ) in addition to gripper length.

Because the RACU strictly uses integers and does not accept floating point, Angles must be converted from a regular angle system to a RACU number, $n_{joint}$ , discussed in Subsection II-D using equations 1 to 5.

The following functions are developed for articulate positioning:

Pos = Go_Robot(Obj, Grp, Grp_Rot, Wrist, Arm, Shldr, Base). The function conducts movement to designated position using Articulate system representation. Input parameters include: Obj: Object reference to serial port, Grp: Gripper length, Grp_Rot: Gripper Rotation, Wrist: Wrist orientation ( $\theta_{wrist}$ ), Arm: Arm orientation ( $\theta_{elbow}$ ), Shldr: Shoulder orientation ( $\theta_{shldr}$ ), and Base: Base orientation ( $\theta_{base}$ ). The return value is the final position after movement.

Pos = Get_Degree_Robot(Obj). This function gets position in world coordinates. Input parameter is Obj: Object reference to serial port. The return value is the position in regular world articulate system.

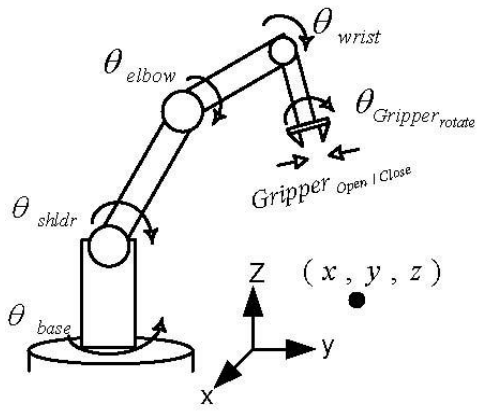


Figure 4 Robotic Arm Degree of Freedom.

Pos = Go_Robot_Block(Obj, Grp, Grp_Rot, Wrist, Arm, Shldr, Base). Conduct movement to designated position using Articulate system, in addition, block Matlab until Robotic Arm reaches designated position. Input parameters include Obj: Object reference to serial port, Grp: Gripper length, Grp_Rot: Gripper Rotation, Wrist: Wrist orientation ( $\theta_{wrist}$ ), Arm: Arm orientation ( $\theta_{elbow}$ ), Shldr: Shoulder orientation ( $\theta_{shldr}$ ), and Base: Base orientation ( $\theta_{base}$ ). The return value is the final position after movement.

Pos = Stop_Robot(Obj). This function stops any current movement the Robotic Arm is conducting. It has one input parameter, which is Obj: Object reference to serial port. The return value is final position of Robotic Arm in regular articulate system.

Wait_for_Robot(Obj). This function checks whether arm is conducting movement. If yes, block Matlab and force user to wait until arm reaches designated position. It has one input parameters, which is Obj: Object reference to serial port. The return value is final position of Robotic Arm in regular articulate system.

Pos = Move_Joint_Robot(Obj, Jnt, Dir). The function moves one joint of the robot until a stop order is initiated or arm reaches its physical limit. Input parameters include Obj: Object reference to serial port, Jnt: Reference to joint number,( as in Table 3 ) and Dir: Direction of movement (-1: Left/Down/Close, and 1: Right/Up/Open). The return value is the final position of Robotic Arm in regular articulate system.

2) Cartesian System: A coordinate system that uses Cartesian dimensions (X,Y ,Z) as the bases for positions around Robotic Arm in space. As a result, any target position is coded using three values: x, y, and z.

For the RACU, angles of Robotic joints need to be computed to put the arm gripper at specific (x, y, z). However, this computation produces infinite number of solutions as the number of unknowns is greater than the number of known values. The challenge of solving such ill-conditioned problem is beyond the scope of this paper. One option to relax the problem is using Fuzzy Inference System (FIS), [5], to less the number of unknowns. The library introduced in this paper includes a number of functions that deal with such coordinate system, which are presented below.

Pos = Go_Cart_Robot( Fis, obj, x, y, z, pref) is a movement function that orders the robot to reach a target position, (x, y, z), using the Cartesian coordinate system. The input parameters include Fis: reference to location of a fuzzy system that helps in solving the ill-conditioned problem, Obj: object reference to serial port, (x, y, z): target position, Pref: preferred initial $\theta_{wrist}$ to simplify the complexity of the ill-conditioned problem. For example, an application where the arms wrist joint is expected to always point down should have a value of -90◦, such as in Fig. 4. The return value of this function is the final position of the robotic arm.

`Pos = Get_Cart_Robot(Obj)` is a function that gets the position of the arm gripper in Cartesian coordinates. The input parameter is `Obj`: the object of the serial port.

### *D. Robotic Arm Additional Functions*

The RACU has a number of other options that can help enhance the experience of using the Robotic Arm. The emergency button, available as a separate unit in LabVolt system, can be used as a safety measure when pressed. However, the RACU remains locked until it is released by a specific command.

In addition, the servo motors of the Robotic Arm can be torqued or freed based on the application used. For example, if the application needed to deal with weight, it is better to set the Robotic Arm in torque mode. Basically, this mode gives sustained power to the Robotic Arm.

The following functions implement the different modes. Emergency_Robot(Obj) Activates the emergency status as if the physical emergency button is pressed.

Release_Emergency(Obj) releases the RACU to continue work after the emergency button is physically released. If the physical emergency button is still pressed an error message will be returned. Msg = Torque_Robot( Obj) activates torque mode so that heavy weights can be held by the Robotic Arm. The return value is a message from the RACU stating confirming the action is performed. Msg = Free_Robot(Obj) deactivates torque mode and releases the servo motors of the arm. The return value is a confirmation message.

### *E. Robotic Arm Matlab Library Remarks*

The functions mentioned in the library use “printf” for sending and “scanf” for receiving data from the RACU. Both functions are low level I/O functions. For this reason, caution should be taken when attempting to send commands manually.

In addition, the RACU waits for a line terminator “0D” when sending data, which can be specified in Matlab as an ending bit. However, when the RACU sends data the receiving user Matlab in this case should look for “0A” line terminator.

By trial it is found that Matlab cannot change between terminators after specifying the terminator during initialization and receiving a reference to the serial port, due to the asynchronous communication with the serial port. This issue has been solved in this work by designing a condition statement that waits and checks if data is received in the Input Buffer.

Also, If a new command is sent while Robotic Arm is busy conducting an operation, RACU returns a busy signal “BSY”. There is a chance that the RACU crashes if unknown commands are sent, or a command is sent while busy.

## IV. CONCLUSION

The Robotic Arm Controller Device was successfully interfaced and the Robotic Arm can be used independently without the need for RoboCIM.

In addition, by using the decoded low level commands, the RACU can be interfaced into any system, with any programming language, such as C++. The only prerequisite is to setup a successful serial connection with the specified parameters and follow protocol rules mentioned in the paper.

Finally, for this work a Matlab library is developed and ready for use. The functions in the library will handle all the low-level tasks in terms of communication with the RACU, control and exception handling. this tool will give developers the opportunity to focus on more abstract higher level problems.

Delays in the response of RACU to the high-level commands continue to open new challenges that will be investigated to further enhance the Robotic Arm open source library.

## REFERENCES

[1] electronicsforu.com, http://electronicsforu.com/technology-trends/open-hardware-sparks-innovation-robotics, accessed: 15/Mar/2017.

[2] LapVolt systems, MECHATRONICS and INDUSTRIAL MANUFACTURING TECHNOLOGY.

[3] engineersgarage.com, http://www.engineersgarage.com/articles/what-is-rs232, accessed: 15/Mar/2017.

[4] M. Elsharif, https://github.com/Moejoe90/LabVolt-5250-Library, accessed: 15/Mar/2017.

[5] L. Zadeh, “Fuzzy Sets*,” Information and control, vol. 8, no. 3, pp. 338–353, 1965.

## BIOGRAPHIES

**Mohamed Elsharif** holds B.Sc. degree in Computer Engineering from University of Tripoli. He is currently working as a developer for a start-up Shnaboo, that looks into integrating Augmented Reality solutions. His research interests include Intelligent Embedded Systems, Pattern Analysis and Recognition, Machine Learning and Augmented reality sensor fusion.

**Nabil M. Drawil** holds Ph.D. and M.Sc. degrees in Electrical and Computer Engineering - University of Waterloo, Canada, and B.Sc. degree in Computer Engineering - University of Tripoli. He is an Assistant Professor with the department of Computer Engineering, University of Tripoli. His research interests include Intelligent Systems, and PAMI.

**Mahmoud Osman** holds a Ph.D. degree in Control Engineering from the University of Cairo, Eygept, M.Eng in Control Systems from University of Shefield, UK, and B.Eng (honours) in Electrical and Electronic Engineering from Cardiff University, UK. He is currently a Professor with the Department of Computer Engineering, University of Tripoli. His research interests include Adaptive Control and Intelligent Control Systems.