Real Time Distributed Controller For Delta Robots

Ali Sharida, Iyad Hashlamon
Mechanical Engineering Department
Palestine Polytechnic University
Hebron, Palestine
156049@ppu.edu.ps, iyad@ppu.edu

Abstract: - This paper investigates a real time distributed controller for a 3 DOF delta robot using low-cost educational simple microcontrollers. The parallel computing technique is used were the computational load is divided among several microcontrollers networked to each other to implement control methods. More specifically, the computation is distributed among four microcontrollers (MCU's) which are connected to each other using CAN bus protocol. The main MCU is used to compute the control law. Each of the remaining MCU's is connected to one actuator and its attached encoder to form an Intelligent sensor-actuator system (ISAS). At each sample time, the ISAS broadcast a message using the CAN bus to the main MCU containing the information about the motor position. Then according to the control law, ISAS receives the corresponding controller value that has to be applied to the motor. All required periodic, aperiodic and sporadic tasks, were implemented and will be handled by these MCU's. Using this design, the computation time of control law can be minimized and implemented using ARDUINO microcontrollers. More, this method increases the flexibility of the system for additional equipment and control by adding more nodes to the network. The results show the applicability of the proposed distributed controller, it can track different types of control signals with acceleration up to 9.8 m/s^2 (1g).

Key-Words: - Real Time Control, Distributed Control, Delta Robot.

1 Introduction

Delta robots are widely used in applications that require very fast motion and accuracy, such as picking and placing [1, 2]. The main advantage of these robots is the ability to produce high acceleration at the end effector. Furthermore, as the mass of the overall system is relatively low, these robots can achieve a high load capacity. Therefore, this robot attracted many researchers to develop kinematic and dynamic models and controllers [3-7].

However, delta robots control contains multiple computational tasks that should be completed in terms of modeling, planning and control [8], these tasks require relatively long computational time. So, it is very important to use a method to minimize computational time in such applications. Furthermore, the robot is equipped by 3 actuators and 3 encoders, 1 encoder per actuator. Then, the microcontroller should deal with 6 (2 channels for each encoder) channels that generate digital pulses with high frequency. Thus, hardware problems appear low cost educational microcontrollers are used for delta robots. In general, the frequency of these pulses is very high due to the high speed motion. Missing any of these pulses results in an incremental error of measuring the angular position of the actuated joints which is accumulated with time.

The comparative study in [9] illustrates the processing time of the fundamental three approaches of modeling Delta robot "Principle of Virtual Work, the Newton-Euler Formulation, and the Lagrangian Formulation). Although the results were: Principle of Virtual Work requires 0.73 sec, Newton-Euler Formulation requires 1 sec, and Lagrangian Formulation requires 0.37 sec, these processing times are high for real time systems when model-based control approaches are used and the model is tuned online adaptively. The challenge becomes harder when the control law and the reading from the sensors are considered.

To overcome the aforementioned challenges, distributed control approaches were used. In these approaches, several microcontrollers (MCU's) are used, each controller is assigned its own job(s). The microcontrollers communicate with each other using a communication protocol to form an overall real time Network Control System NCS.

Many protocols can be used to establish a real time network, such as I2C [10], SPI [11] and CAN protocol [12]. Among them, CAN protocol has many advantages including very simple physical construction, it supports auto retransmission of lost massages and supports different error detection capabilities. Therefore, it is considered the most suitable communication method for real time applications.

A distributed controller is reported in [13], where the controller is designed to control a slave robot from a master arm using SPI protocol. In [14], the researcher implemented an embedded controller for 5 DOF manipulator using SPI protocol using a simple PID controller. Although SPI is a very simple protocol and depends on the principle of master and slave communication, any fault in the master MCU will lead to shut down all network. Furthermore, SPI communication requires more signal lines than other communication protocols, which increase the complexity of the network.

To solve this problem, the principle of parallel computing should be employed [15], where the tasks of data acquisition and computing control law are distributed on multiple controllers. This ensures that the frequency of the controller is greater than the frequency of controlled system, provides the advantage of minimizing computational time and increases the flexibility of adding new tasks (nodes) or editing the existing ones.

In [16] a distributed CAN-Based Architecture for hardware control and sensor data integration was proposed for a mobile robot platform. However, the algorithm was implemented for general purpose computers. In the same context, in [17], a CAN bus based distributed controller was designed to control a mobile robot for picking and placing. It employed the principle of parallel processing to perform the functions of obstacle avoidance, driving, path planning and inspection.

This paper proposes a real time control for the 3-DOF delta robot. It uses four Microcontroller Units (MCU), each one consists of a microcontroller and a Controller Area Network bus (CAN bus) receiver-transmitter. One MCU is used to compute the control law. The other three MCU's are connected to the actuated joints through an electronic interfacing module, each one of the three MCU's along with the actuator and sensory system forms an intelligent sensor-actuator-system ISAS.

Each ISAS is connected to one actuator and one sensor. Further, it can communicate with other ISAS's and the controller MCU through CAN bus communication protocol. The ISAS reads the actuator position through an encoder, forms the necessary signal processing and prepares the ready measured data in a massage and broadcasts it to the CAN bus. This massage will be received by the beneficiary MCU, and in the same way for all ISAS's.

The controller MCU computes the required control law and broadcasts it on the CAN bus. Each ISAS will receive its own massage and skip the others. Then each ISAS analyses the massage and applies the required signal on the actuator. This approach enhances flexibility to the system for changing the control approach and adding other jobs by adding new nodes to the network containing the desired tasks and jobs, these additional nodes can be used for applications such as vision control. Adding a new node will not change the physical structure of the distributed controller, since each ISAS and the main MCU controller will remain the same, the added node will be used to supply the controller with required information related to the new function. Further, distributing the computational load among 4 MCU's minimizes the sampling time which in turn increases the stability and accuracy of the system. The work is simulated using MATLAB with the TrueTime toolbox [18] and implemented practically using ARDUINO microcontroller. The overall block diagram of this approach is shown in Fig.1.

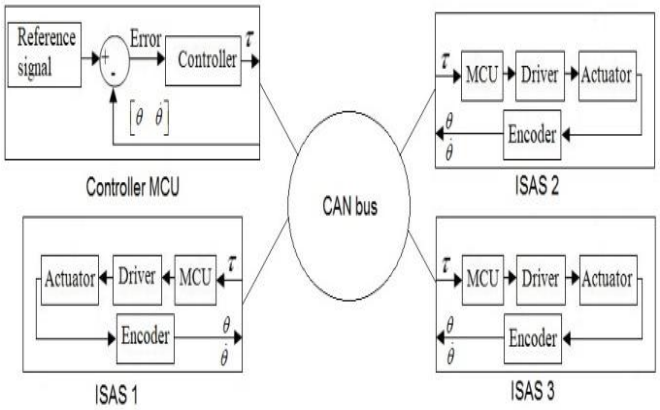

Figure 1. Real time network structure.

2 Real time system design

The design of real time Network Control system (NCS) starts by assigning the tasks with their timing constraints. For this system there are three motors to be actuated and controlled. In order to perform the control law, the following data is required: the reference signal or trajectory that the end effector must track, the forward kinematics model to transform the joint space variables to the task space variables, inverse kinematic model to transform the task space variables to joint space variables, the robot dynamic and inverse dynamic model to compute the control signal, a controller, and some computations in addition to sending and receiving data through the CAN bus as shown in table 1. The control process is divided into 12 different tasks as follows:

Task 1: it is responsible to compute the angular position and velocity of each actuator from the signal measured from an encoder attached to the actuator. This task is sporadic as the position should be correctly computed within its time limits, any delay in this task will cause drifting in position computation, which will lead to instability of the entire system.

Task 2: it is responsible to read the desired inputs from the user. This task is aperiodic task, as the user will not apply these inputs periodically. Furthermore, there is no matter if these inputs were used to compute control law in the next cycle.

Task 3: it is a periodic task, it has the jobs which perform the forward kinematics. This task depends on the results from Task 1. It takes the measured angles as inputs, and then it computes the related position of the end effector. This task should be executed on each program cycle, so it is a periodic task (Task 4 to 9 are periodic for the same reason).

Task 4: In this task, the inverse kinematics is computed, which will be used later to generate the required feedback variables for the controller.

Task 5: Jacobian matrix performs a transformation of the velocities from joint space to work space. This task is responsible to compute this matrix in order to compute the velocity of the end effector.

Task 6: In this task, the inverse of Jacobian will be computed, in order to get the required angular velocities of the actuators that required to control the end effector.

Tasks 7 and 8: These tasks are responsible of computing the required actuators torques to control the motion of the end effector.

Task 9: In this task, a trajectory is designed based on the received reference signal to ensure that the motion of the end effector is smooth.

Tasks 10 and 11: These tasks are responsible of sharing data among the MCU's. They are an on change based tasks, which will be enabled when the current position or the computed torque is changed. They should be executed directly when their flags are enabled to ensure that the control law will be computed correctly on any change of links kinematics.

Task 12: This is an aperiodic task, it computes the related voltage of the resulted torque signal. It will be executed when the computed torque is changed. It will apply torque signal to the actuator by computing the equivalent voltage and applying it to the actuator.

Table 1. Tasks time constraints.

Task Name	Type	Execution Time (ms)	Period= dead line Time (ms)
Compute system states	Sporadic	0.1	8.5
Read reference signal	Aperiodic	0.3	8.5
Forward Kinematics	Periodic	0.9	8.5
Inverse Kinematics	Periodic	0.4	8.5
Jacobian	Periodic	0.6	8.5
Inverse Jacobian	Periodic	0.9	8.5
Dynamics	Periodic	0.8	8.5
Inverse Dynamics	Periodic	1.1	8.5
Trajectory	Periodic	0.2	8.5
Data transmission (CAN bus)	Sporadic	0.6	8.5
Data reception (CAN bus)	Sporadic	0.7	8.5
Apply outputs	Aperiodic	0.1	8.5
Total execution time			6.7 ms

The scheduling of the tasks is based on their type and timing. The periodic tasks were scheduled using the fixed priority algorithm Rate monotonic (RM). Whereas the aperiodic and sporadic tasks were scheduled by the principle of servers. In real time systems multiple servers can be used to handle the non-periodic tasks such as Bandwidth-preserving, periodic and sporadic servers [19]. In this project the non-periodic tasks were classified into two types; Event based tasks that were executed using sporadic server and on change based tasks that were executed using periodic server.

An Event based task enables a flag when it is released. This flag can be assumed to be a global flag that can be noticed in any part of the algorithm and can directly pre-empt the current executing task and jumps to a service routine. This type of tasks will be executed directly when its flag is on, and the microcontroller will interrupt any executing task in this case. Any event based task was assumed to be

non-pre-emptible and will continue executing until it is finished. For example, task 1 represents an event based task, as each encoder is connected to microcontroller's external interrupt. This interrupt will be enabled directly when the encoder generates any signal.

An on change based task contains a flag that will be turned on when the task is released. However, this flag should be tested periodically to determine readiness the task with its resources. Accordingly, the algorithm decides if an interrupt is required or not. These tasks have higher priority than the periodic tasks but less than the event based tasks. This type was handled using Periodic Server algorithm which creates a periodic server that is responsible of checking the flag of the task. If the task is released, it will be executed, while if it is not released, the server will be pre-empted until the next period.

3 Schedulability test

The timing constraints are used the schedulability test. Namely the execution time and deadline of each task. The execution time of each task was computed experimentally using oscilloscope, each task was implemented individually, a hardware flag was turned on at the beginning of the execution and turned off at its end, then the HIGH-interval was captured. The worst case execution time (WCET) is considered as the time required to complete all the tasks if they were released at the same instant and their recourses are available without violating their constraints . According to table 1, the WCET is 6.7 ms. To avoid processor over loading, the period was adjusted to 8.5 ms, this time was assumed to be the deadline for all tasks in the current cycle, and the released time for the next cycle. Each periodic task is released at the beginning of the period, these tasks should be executed before the deadline which presents the end of current period.

In this project, the sampling time was selected to be 8.5 ms. As the controller is distributed, all tasks will be executed before the deadline, as the worst case execution time occurs at the computation of control law which requires 6.5 ms, it is schedulable based on RM utilization factor test, as the utilization factor is about 76%. RM algorithm is used here because it is schedulable for identical parallel Microcontrollers, where the execution time is the same for a task if it is schedulable for identical parallel Microcontrollers, where the execution time is the same for a task if it is executed on any processor [20]. Moreover, Fig. 2 shows that RM can produce a feasible schedule for the system at WCET. The shown schedule depends on the parameters listed in Table 1. At the beginning of the period (at t=0), all periodic tasks are released at the same time, while the deadline of all tasks is located at the end of the current period.

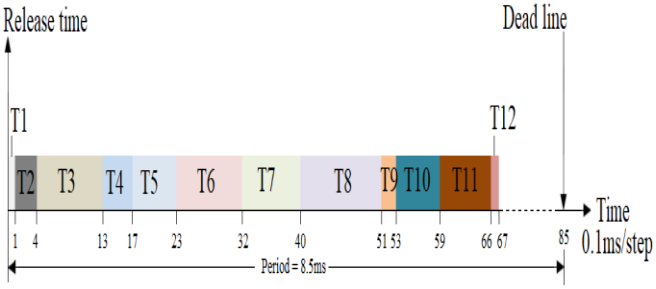

Figure 2. Tasks scheduling based on RM.

4 Data handling and frames creation

The previous tasks were distributed on 4 MCUs, each MCU represents a single node on the network as shown in Fig. 3. The periodic tasks that are required to compute the control law were implemented on the controller MCU. Therefore, this node will not deal with the outputs or inputs to the actuators. At each program cycle, it will create a CAN message frame at the end of this cycle, this frame contains 6 bytes of data, each element of torque vector τ is presented using 2 bytes, the 1st Byte represents the direction of the torque, the other one represents the absolute value of the torque element as shown in Fig. 4. For the other nodes, each one is connected to a single actuator and its digital sensor. The digital encoder consists of 2 digital channels, these channels will continuously change their values as the actuator is running. Therefore, each MCU should enable the external interrupt to compute the equivalent actuator's joint position θ and velocity $\dot{\theta}$ states. Whenever one of the states is changed, the MCU will directly create a CAN message with data length of 4 bytes as shown in Fig. 5.

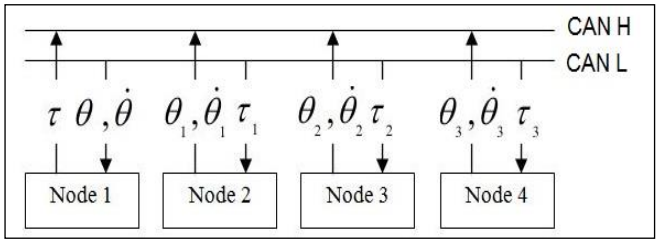

Figure 3. System network.

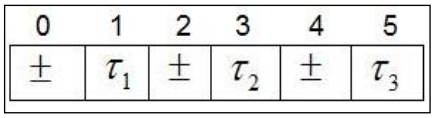

Figure 4. Control law data frame.

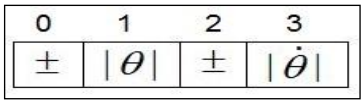

Figure 5. States data frame

5 Real time distributed control algorithm (RTDCA)

The last step of implementing the proposed approach is to develop an algorithm that satisfies the scheduling requirements proposed previously. The algorithm was developed using C programming language as it provides the least computational time compared with other programming languages (except assembly language). RTDCA consists of 2 parts, the first one is the high level processing and control stage, it is executed by the main controller. The second part is composed of computing the angular position and velocity of the actuated joints and applying control signal on the actuators. It is executed by the other three ISAS's.

The tasks of the former part are executed periodically at each program cycle. However, it contains aperiodic and sporadic tasks. As any of the sporadic interrupts is enabled, the algorithm will pre-empt the running task and start executing the sporadic one. For the aperiodic tasks, they were handled by the mean of periodic non preserving scheduling servers, such that, the aperiodic task will be checked periodically. If it is released, it will be executed, else, it will be pre-empted until the next cycle, these steps were illustrated in the flowchart in Fig.6.

Main controller Pseudo code:

1. Enable interrupts (timer 0 interrupt and external interrupt).
2. Initialize variables, Initialize CAN-BUS.
3. Check interrupt flag
   - If interrupt flag is HIGH
     - Pre-empt current task, and execute the sporadic task.
   - Else
     - Continue.
4. Compute position kinematics.
5. Compute velocity kinematics.
6. Compute next position.
7. Compute error.
8. Compute control law.
9. Create and decode CAN frame.
10. Transmit frame.
11. Check CAN-BUS for received frames.
    - If there is new frame
      - Receive frame.
      - Decode frame.
      - Update states.
    - Else
      - Continue.
12. Go to step 4.

In the later part, each of the other three ISAS's checks periodically if there is a new frame. If yes, it will receive the new frame then decodes the message to extract the control law output. Then it computes the control torque equivalent voltage in order to apply it to the driver. Each MCU has a hardware ID which is formed at the initialization level, this ID consists of 2 binary bits represented by 2 fixed hardware digital inputs. The ID of the first MCU is (01)B. The second and third MCU's are assigned to ID's (10)B and (11)B respectively. If the acquired states by the encoder are changed, the MCU will directly create CAN frame which consists of its ID and the values of the states as described previously. These steps are described in the flowchart in Fig.7 along with the following Pseudo code:

1. Enable interrupts (timer 0 interrupt).
2. Initialize variables, initialize CAN-BUS.
3. Check CAN-BUS for received frames.
   - If there is new frame
     - Receive frame.
     - Decode frame.
     - Compute required voltage.
     - Apply voltage on actuator through the driver.
   - Else
     - Continue.
4. Read actuator states
   - If new states do no equal previous states
     - Encode new states.
     - Create CAN frame.
     - Assign MCU's ID to the frame.
     - Transmit frame.
   - Else
     - Continue.
5. Update states.
6. Go to step 4.

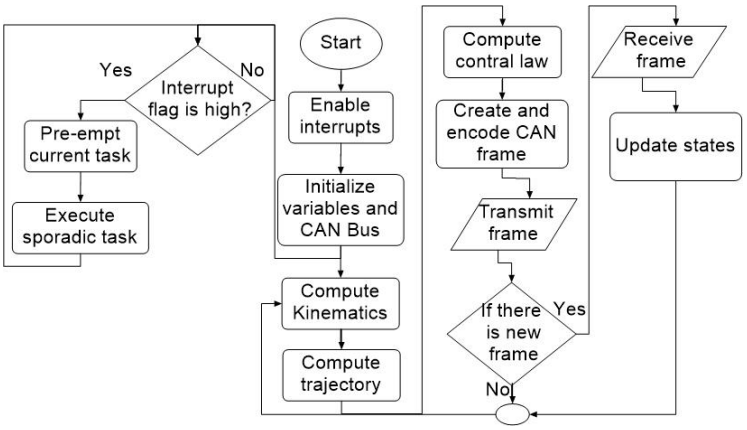

Figure 6. Main controller algorithm.

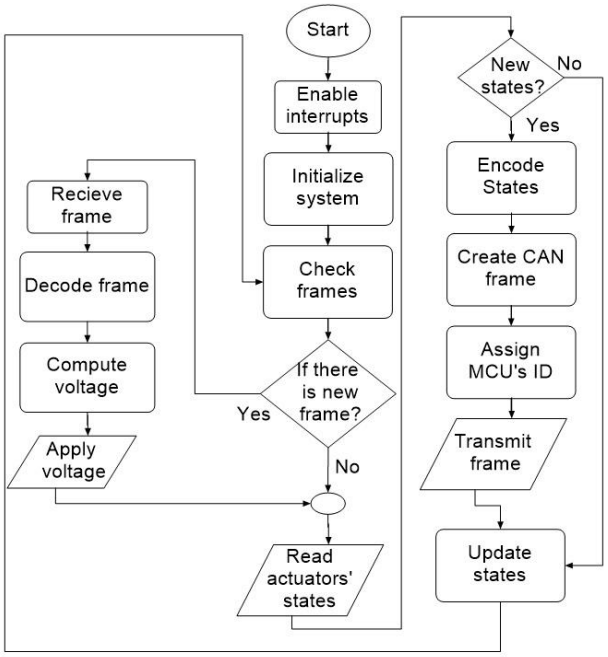

Figure 7. Symmetrical ISAS algorithm.

6 Results

The system was implemented and simulated using MATLAB, the design of the controller MCU and ISAS 1 are shown in Fig.8, and 9 respectively. The controller MCU receives angles frame from other MCU's, each frame contains data that represent the amplitude and the direction of the angular position along with the ID of the transmitter MCU. Also it receives the desired position as a vector of the desired coordinates [X ; Y ; Z]. This signal is the reference signal. After that, MCU controller computes the required actuator's torque and creates a frame of torque data to broadcasts it through CAN bus.

Figure 8. controller MCU design.

The other 3 MCU's are symmetrical, each one is physically connected to an actuator and a position incremental encoder. The measured state θ and pseudo measured state $\dot{\theta}$ will be transmitted to the controller MCU by a frame that contains joint's information and the ID's of the receiver and transmitter (1:2 means that the MCU with ID 2 will transmit frame to MCU with ID 1). The simulation is carried on using the TrueTime toolbox.

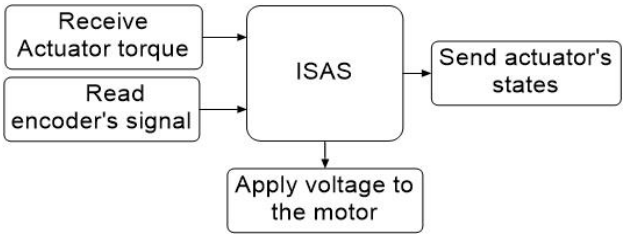

Figure 9. ISAS 1 design.

The algorithm was implemented on a hardware delta robot shown in Fig.10.a with parameters shown in Fig.10.b listed in Table 2.

The system was controlled using PD inverse dynamics (computed torque) algorithm as described in [21], the control law can be written in the form,

$$\tau = M(\theta)\left[\ddot{\theta}_d + K_d\dot{e} + K_p e\right] + C(\theta,\dot{\theta})\dot{\theta} + G(\theta) \quad (1)$$

Where τ is 3×1 control vector, $M(\theta)$ is 3×3 mas matrix, $\ddot{\theta}_d$ is 3×1 desired angular acceleration, $\dot{e}$ is angular speed tracking error, e is position tracking error, $C(\theta,\dot{\theta})$ is 3×3 coriolus matrix, $G(\theta)$ is 3×1 gravity vector, $\ddot{\theta},\dot{\theta}$ and θ are the angular acceleration, velocity and position of the actuated joints, and K_p, K_d are controller gains,

$$K_p = K_d = \begin{bmatrix} 50 & 0 & 0 \\ 0 & 50 & 0 \\ 0 & 0 & 50 \end{bmatrix} \quad (2)$$

As shown in Fig. 11, the response of the robot tracks the reference signal [0.1; -0.1; 0.5]. The results show that the response of the robot is stable and can achieve its tasks in the desired manner using the specified trajectory time. While Fig.12 shows the required torque on each actuated joint to move the end effector from its initial to final position [0.1; -0.1; 0.5] as was planned. Finally, Fig.13 shows tracking error on each axis which converges to zero during 0.5s.

Table 2. Delta robot parameters

Parameter	Value
f	0.1 m
e	0.055 m
L_a	0.18 m
L_b	0.435 m
Mass of moving platform	0.196 kg
Mass of elbow	0.024 kg
Mass of the forearm	0.055 kg
Mass of upper arm	0.190 kg
Motor inertia	81.6×10^{-3}
Motor gear ratio constant	0.01

Figure 10.a. Delta robot prototype

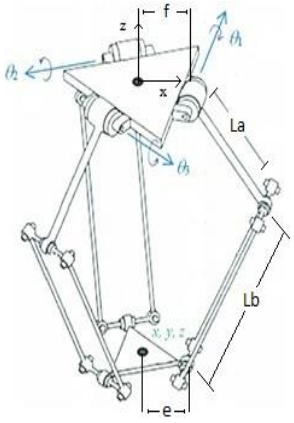

Figure 10.b. Robot kinematic diagram

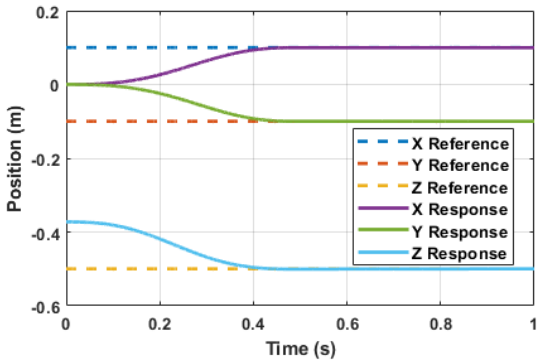

Figure 11. Robot response for (0.1, -0.1, 0.5) reference signal.

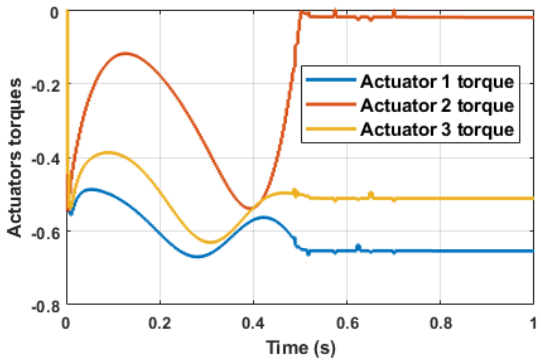

Figure 12. Actuated joints torque.

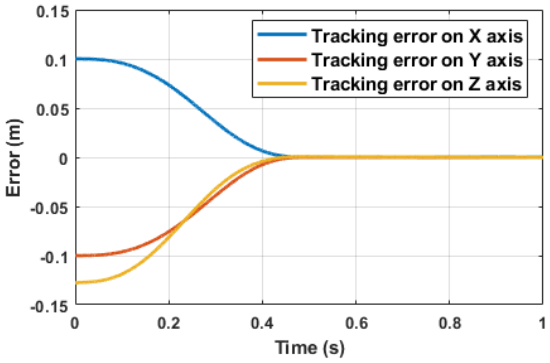

Figure13. Tracking error.

Moreover, a spiral reference signal was applied as shown in Fig.14, the result shows that the distributed controller can track not only a step constant input, but also a spiral reference signal, where the desired position changes continuously with time.

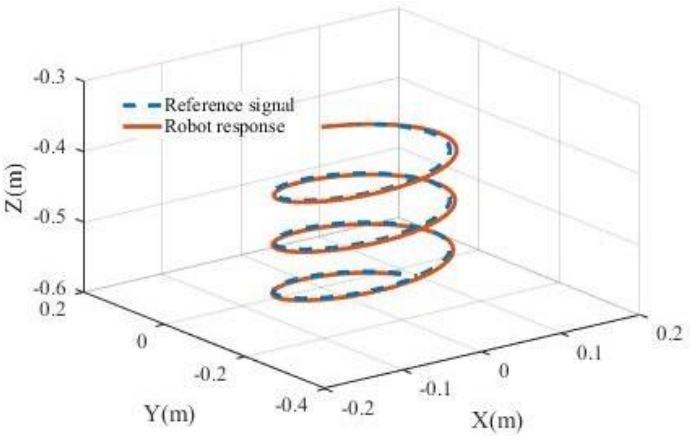

Figure 14. Tracking spiral signal.

7 Conclusion

In this project, a real time distributed controller was implemented for delta robots. The main contribution of this paper is the design of a real time flexible distributed control network using CAN Bus protocol. This distributed controller minimizes the sampling time and makes it more stable and applicable for low cost microcontrollers such as ARDUINO and PIC. More, the proposed distributed controller is very flexible due to the ability to add additional nodes for future features and new functions. This design was able to handle and execute all periodic, aperiodic and sporadic tasks efficiently without any missing in their deadlines. The response of the robot was stable and rapid. Using this approach, the computation time was minimized as it was distributed on 4 MCU's. This design can be applied for any type of manipulators. Furthermore, this approach allows sampling time to be reduced to 6.5 ms, while the sampling time was adjusted to 8.5 ms to avoid processor overloading.

References

[1] J. Brinker, N. Funk, P. Ingenlath, Y. Takeda, B. Corves, Comparative study of serial-parallel delta robots with full orientation capabilities, *IEEE Robotics Automation Letters*, 2 (2017) 920-926.

[2] M. Rachedi, B. Hemici, M. Bouri, Design of an H∞ controller for the Delta robot: experimental results, *Advanced Robotics*, 29 (2015) 1165-1181.

[3] P. Bai, J. Mei, T. Huang, D.G. Chetwynd, Kinematic calibration of Delta robot using distance measurements, *Journal of Mechanical Engineering Science*, 230 (2016) 414-424.

[4] Y.-L. Kuo, P.-Y. Huang, Experimental and simulation studies of motion control of a Delta robot using a model-based approach, *International Journal of Advanced Robotic Systems*, 14 (2017).

[5] C. Wang, Y. Fang, S. Guo, Multi-objective optimization of a parallel ankle rehabilitation robot using modified differential evolution algorithm, *Chinese Journal of Mechanical Engineering*, 28 (2015) 702-715.

[6] H.-Q. Zhang, H.-R. Fang, B.-S. Jiang, S.-G. Wang, Dynamic performance evaluation of a redundantly actuated and over-constrained parallel manipulator, *International Journal of Automation Computing*, 16 (2019) 274-285.

[7] O. Ibrahim, W. Khalil, Inverse and direct dynamic models of hybrid robots, *Mechanism machine theory* 45 (2010) 627-640.

[8] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, Robotics: modelling, planning and control, Springer Publishing Company, 2010.

[9] J. Brinker, B. Corves, M. Wahle, A comparative study of inverse dynamics based on clavel's delta robot, in: *Proceedings of the 14th World Congress in Mechanism and Machine Science.* , Taipei, Taiwan, 2015, pp. 25-30.

[10] J. Sastry, J.V. Ganesh, J.S. Bhanu, I2C based networking for implementing heterogeneous microcontroller based distributed embedded systems, *Indian Journal of Science and Technology*, 8 (2015) 1-10.

[11] J.J.R. Raj, S. Rahman, S.J.E.s. Anand, a.i.j. technology, 8051 microcontroller to FPGA and ADC interface design for high speed parallel processing systems–Application in ultrasound scanners, *Engineering science technology, an international journal*, 19 (2016) 1416-1423.

[12] A. Ashiebi, A. Khalil, J. Wang, Networked control of parallel DC/DC converters over CAN bus, in: *IEEE International Conference on Power System Technology (POWERCON)*, IEEE, 2016, pp. 1-6.

[13] I. Jaziri, L. Chaarabi, K. Jelassi, A remote DC motor control using Embedded Linux and FPGA, in: *7th International Conference on Modelling, Identification and Control (ICMIC)*, IEEE, 2015, pp. 1-5.

[14] C. Urrea, J. Kern, Development of an electronic controller applied to a robotized manipulator, *Computers Electrical Engineering*, 56 (2016) 648-658.

[15] D. Henrich, J. Karl, H. Wörn, A review of parallel processing approaches to robot kinematics and jacobian, *Technische Universität Kaiserslautern*, (1997).

[16] D.P. Losada, J.L. Fernández, E. Paz, R.J.S. Sanz, Distributed and modular CAN-based architecture for hardware control and sensor data integration, *Sensors*, 17 (2017) 1-17.

[17] R. Chen, B. Liu, M. Pan, H. Zhou, Design of Distributed Control System for the Pick-up Robot Based on CAN Bus, in: *IEEE International Conference on Mechatronics and Automation (ICMA)*, IEEE, Tianjin, China, 2019, pp. 102-107.

[18] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, K.-E. Arzen, How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime, *IEEE control systems magazine*, 23 (2003) 16-30.

[19] F. Liu, A. Narayanan, Q. Bai, Real-time systems, Prentice Hall PTR, United States, 2000.

[20] J.G. S.K Baruah, Rate-monotonic scheduling on uniform multiprocessors, *IEEE Transactions on Computers*, 52 (2003) 966-970.

[21] F.L. Lewis, D.M. Dawson, C.T. Abdallah, Robot manipulator control: theory and practice, CRC Press, 2003.