Evaluation of Sequential and Parallel Strategies for IRB 140 Dynamics through Symbolic Derivation and Numerical Simulation

Naisha Kishore
Department of Mechanical Engineering
National University of Technology Karnataka, Surathkal
Mangalore, Karnataka
kishorenaisha1@gmail.com

Aneesha Deshmukh
Department of Mechanical Engineering
National University of Technology Karnataka, Surathkal
Mangalore, Karnataka
anudeshmukh0207@gmail.com

Kundan Kanti Saha
Department of Information Technology
National University of Technology Karnataka, Surathkal
Mangalore, Karnataka
kundansaha@gmail.com

***Abstract*—This paper presents a comparative study of computational strategies for the dynamic modeling and simulation of the ABB IRB 140 6-DOF robot within MATLAB. We begin with the sequential symbolic derivation of the dynamic equations using the recursive Newton-Euler algorithm, establishing a baseline. We then explore parallel accelerations for this symbolic phase using both distributed memory (MPI) and shared memory (parfor, akin to OpenMP) approaches. These methods yield symbolic expressions for the mass matrix, Coriolis/centrifugal terms, and gravity vector. Performance characteristics and trade-offs associated with sequential, distributed memory and shared memory are discussed in the context of robot dynamic analysis.**

***Index Terms*—Robotics, Dynamic Modeling, Simulation, IRB 140, Newton-Euler, Sequential Computation, Parallel Computing, MPI, OpenMP, parfor, Symbolic Computation, Performance Analysis**

I. Introduction

Dynamic modeling plays a crucial role in modern robotics, enabling advanced control strategies, accurate simulation, and trajectory optimization. For industrial manipulators like the ABB IRB 140, precise dynamic models are essential for achieving high-performance motion control, especially in high-speed or high-precision tasks. However, this modeling process imposes significant computational demands, particularly at two distinct phases: symbolic derivation and numerical simulation.

The symbolic derivation phase involves generating the analytical expressions that represent the robot's dynamics. This process is computationally intensive due to the complexity of expressions that grow exponentially with the number of degrees of freedom. For a 6-DOF robot like the IRB 140, these expressions can involve thousands of trigonometric terms and operations. While this derivation is typically performed offline, the computational cost can be prohibitive when developing or refining models.

The numerical simulation phase, which applies these derived models repeatedly for analysis or control, presents different computational challenges. Here, real-time or faster-than-real-time performance is often required, necessitating efficient evaluation of complex mathematical expressions thousands or millions of times. As simulation time steps decrease and model fidelity increases, the computational burden grows substantially.

The Recursive Newton-Euler Algorithm (RNEA) provides an efficient approach for computing the inverse dynamics of articulated rigid body systems. This algorithm consists of two recursions: a forward recursion that computes the kinematics (velocities and accelerations) from base to end-effector, and a backward recursion that calculates the forces and torques from end-effector to base. The RNEA enables us to express the standard rigid-body dynamics equation:

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \tag{1}$$

Each component of this equation can be derived from the RNEA by appropriate manipulation and substitution of terms, allowing for a structured approach to capturing the full dynamics of complex robotic systems.

This paper systematically explores various computational strategies for robot dynamics, progressing from fundamental sequential methods to advanced parallel approaches. We begin by establishing a sequential baseline for symbolic derivation using MATLAB's Symbolic Math Toolbox, implementing the RNEA algorithm to derive the complete dynamic model. This serves as our reference point for both correctness verification and performance comparison.

We then investigate two distinct parallel approaches for accelerating the symbolic derivation phase: distributed memory parallelism using Message Passing Interface (MPI), which

enables computation across multiple machines or nodes, and shared memory parallelism using MATLAB's parfor construct (conceptually similar to OpenMP), which leverages multi-core processing on a single machine. These approaches target different aspects of the symbolic computation challenge, with different hardware requirements and scaling characteristics.

Through this progression, we provide a comprehensive evaluation of the computational strategies available for robot dynamics across the full modeling and simulation pipeline. Fig.1 shows the robot's structure and articulation points, helping to visualize the six degrees of freedom that enable its precise movement capabilities in manufacturing and automation applications.

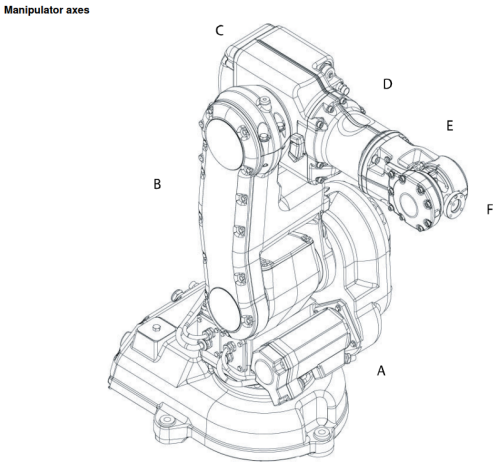

Fig. 1: Degree of Freedom in ABB IRB 140.

II. IRB 140 Robot Model and Parameters

To ensure a fair and meaningful comparison across all computational methods, we maintain consistent model parameters throughout this study. The ABB IRB 140 robot is modeled using identical Denavit-Hartenberg parameters, link masses, center of mass locations, and inertia tensors across all implementations. This consistency is crucial to validate the accuracy of the parallel approaches compared to the sequential baseline and to make meaningful performance comparisons. Any discrepancies in the resulting dynamics would therefore be attributable to numerical precision differences rather than parameter inconsistencies. Fig. 2 shows engineering diagrams of the ABB IRB 140 robot from three viewpoints (front, side, and top), complete with detailed dimensional measurements in millimeters.

A. DH Parameters

The Denavit-Hartenberg (DH) parameters provide a standardized method for describing the kinematic arrangement of rigid links connected by revolute or prismatic joints. For the IRB 140, we utilize the classic DH convention where each joint i is characterized by four parameters: the link length a_i, link offset d_i, link twist α_i, and joint angle θ_i. These parameters define the transformation between consecutive joint

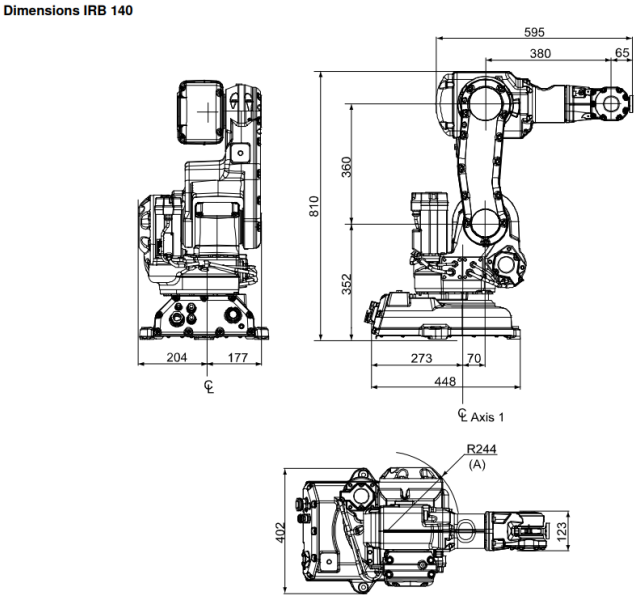

Fig. 2: Dimensions of ABB IRB 140.

coordinate frames, allowing systematic derivation of forward kinematics and, subsequently, the dynamic equations. Table I presents the DH parameters for the IRB 140 robot, where the joint variables q_1 through q_6 represent the controllable degrees of freedom of the manipulator.

TABLE I: DH Parameters for IRB 140

Link i	a_i (m)	d_i (m)	α_i (rad)	θ_i (rad)
1	0	0.352	$\pi/2$	q_1
2	0.070	0	0	q_2
3	0	0	$-\pi/2$	q_3
4	0	0.380	$\pi/2$	q_4
5	0	0	$-\pi/2$	q_5
6	0	0.065	0	q_6

B. Dynamic Parameters

The dynamic behavior of the IRB 140 is characterized by the following physical parameters, maintained consistently across all computational methods:

- **Link Masses (kg):** $m = [9.0, 8.5, 4.8, 1.2, 1.0, 0.5]$.
- **Link CoM Vectors (m):** The center of mass locations are defined in the respective link coordinate frames:

$$^1r_{1c} = [0.025, 0.0, 0.10]^T \quad (2)$$
$$^2r_{2c} = [0.035, 0.0, 0.07]^T \quad (3)$$
$$^3r_{3c} = [0.0, 0.02, 0.05]^T \quad (4)$$
$$^4r_{4c} = [0.0, 0.028, 0.0]^T \quad (5)$$
$$^5r_{5c} = [0.0, 0.0, 0.015]^T \quad (6)$$
$$^6r_{6c} = [0.0, 0.0, 0.012]^T \quad (7)$$

where ${}^{i}r_{ic}$ denotes the position vector from the origin of frame i to the center of mass of link i, expressed in frame i.

- **Inertia Tensors (kg·m^2, about CoM, in frame i):** The inertia tensors are modeled as diagonal matrices for simplicity:

$$\mathbf{I}_1 = \text{diag}(0.1, 0.08, 0.05) \tag{8}$$
$$\mathbf{I}_2 = \text{diag}(0.08, 0.075, 0.06) \tag{9}$$
$$\mathbf{I}_3 = \text{diag}(0.05, 0.04, 0.03) \tag{10}$$
$$\mathbf{I}_4 = \text{diag}(0.015, 0.012, 0.01) \tag{11}$$
$$\mathbf{I}_5 = \text{diag}(0.008, 0.005, 0.008) \tag{12}$$
$$\mathbf{I}_6 = \text{diag}(0.002, 0.0015, 0.001) \tag{13}$$

The gravity vector is defined as $\mathbf{g} = [0, 0, -9.81]^T$ m/s^2 in the base frame, representing standard Earth gravity acting in the negative z-direction. Fig. 3 presents the kinematic structure of the ABB IRB 140 industrial robot. It effectively illustrates the coordinate frames attached to each link (Links 0-3) in the manipulator chain.

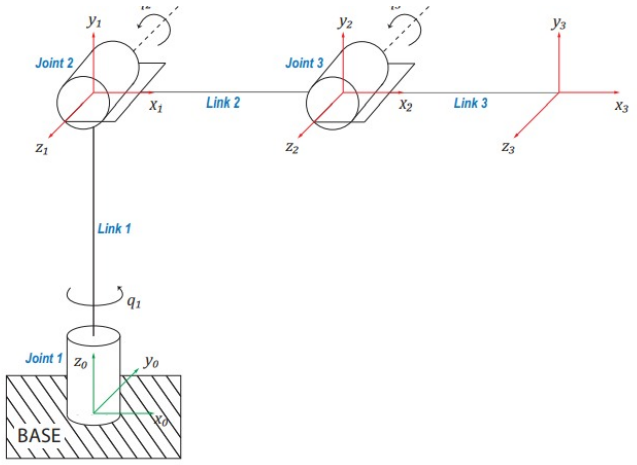

Fig. 3: Links coordinates of ABB IRB 140.

III. Symbolic Derivation Strategies

Our primary objective in the symbolic derivation phase is to obtain closed-form expressions for the mass matrix $\mathbf{M}(\mathbf{q})$, the Coriolis and centrifugal term $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$, and the gravity vector $\mathbf{G}(\mathbf{q})$. We leverage MATLAB's Symbolic Math Toolbox to implement the Recursive Newton-Euler Algorithm (RNEA), allowing manipulation of symbolic variables representing joint positions, velocities, and accelerations. This approach yields exact symbolic formulations that capture the complex relationships between joint variables and dynamic properties. The resulting expressions, while computationally expensive to derive, provide significant advantages for subsequent analysis, control design, and optimization. We investigate three distinct computational strategies for this symbolic derivation phase, each with unique characteristics and performance profiles.

A. Method 1: Sequential Baseline

The sequential implementation serves as our reference approach, performing the symbolic RNEA computation without parallelization. In this baseline method, we first define symbolic variables for the joint positions q, velocities $\dot{q}$, and accelerations $\ddot{q}$. The forward recursion proceeds sequentially from the base (link 0) to the end-effector (link 6), computing transformation matrices, angular velocities, angular accelerations, linear accelerations, and center of mass accelerations for each link. These computations follow the standard RNEA formulation: The forward recursion proceeds sequentially from the base (link 0) to the end-effector (link 6), computing transformation matrices, angular velocities, angular accelerations, linear accelerations, and center of mass accelerations for each link. These computations follow the standard RNEA formulation:

$${}^{i}\omega_i = {}^{i}R_{i-1}\,{}^{i-1}\omega_{i-1} + \dot{q}_i\,{}^{i}z_i \tag{14}$$

$${}^{i}\dot{\omega}_i = {}^{i}R_{i-1}\,{}^{i-1}\dot{\omega}_{i-1} + {}^{i}R_{i-1}\,{}^{i-1}\omega_{i-1} \times \dot{q}_i\,{}^{i}z_i + \ddot{q}_i\,{}^{i}z_i \tag{15}$$

$${}^{i}\dot{v}_i = {}^{i}R_{i-1}({}^{i-1}\dot{v}_{i-1} + {}^{i-1}\dot{\omega}_{i-1} \times {}^{i-1}r_i + {}^{i-1}\omega_{i-1} \times ({}^{i-1}\omega_{i-1} \times {}^{i-1}r_i)) \tag{16}$$

$${}^{i}\dot{v}_{ci} = {}^{i}\dot{v}_i + {}^{i}\dot{\omega}_i \times {}^{i}r_{ci} + {}^{i}\omega_i \times ({}^{i}\omega_i \times {}^{i}r_{ci}) \tag{17}$$

The backward recursion then computes forces and moments from the end-effector to the base:

$${}^{i}F_i = m_i\,{}^{i}\dot{v}_{ci} \tag{18}$$

$${}^{i}N_i = {}^{i}I_{ci}\,{}^{i}\dot{\omega}_i + {}^{i}\omega_i \times ({}^{i}I_{ci}\,{}^{i}\omega_i) \tag{19}$$

$${}^{i}f_i = {}^{i}F_i + {}^{i}R_{i+1}\,{}^{i+1}f_{i+1} \tag{20}$$

$${}^{i}n_i = {}^{i}N_i + {}^{i}R_{i+1}\,{}^{i+1}n_{i+1} + {}^{i}r_{ci} \times {}^{i}F_i + {}^{i}r_{i+1} \times ({}^{i}R_{i+1}\,{}^{i+1}f_{i+1}) \tag{21}$$

$$\tau_i = ({}^{i}n_i)^T\,{}^{i}z_i \tag{22}$$

To extract the components of the dynamic equation, we perform sequential substitutions as shown in Algorithm 1.

This sequential approach, while straightforward to implement, becomes increasingly time-consuming as the robot's degrees of freedom increase, motivating exploration of parallel alternatives.

Algorithm 1 Sequential Extraction of Dynamic Components

// Mass Matrix Calculation
for $i = 1$ to 6 **do**
  for $j = 1$ to 6 **do**
    $\ddot{\mathbf{q}} \leftarrow \mathbf{0}_{6\times 1}$ {Zero vector of length 6}
    $\ddot{q}_j \leftarrow 1$ {Unit acceleration for column j}
    $\mathbf{g} \leftarrow \mathbf{0}_{3\times 1}$ {Zero gravity for mass matrix} {Zero gravity for mass matrix}
    $M(i,j) \leftarrow \tau_{\text{dyn},i}(q, \dot{q}, \ddot{q}, \mathbf{g})$
  end for
end for
// Gravity Vector Calculation
for $i = 1$ to 6 **do**
  $\ddot{\mathbf{q}} \leftarrow \mathbf{0}_{6\times 1}$ {Zero acceleration for gravity}
  $\mathbf{g} \leftarrow [0; 0; -9.81]^T$ {Standard gravity}
  $G(i) \leftarrow \tau_{\text{dyn},i}(q, \dot{q}, \ddot{q}, \mathbf{g})$
end for
// Coriolis and Centrifugal Terms
for $i = 1$ to 6 **do**
  $C\dot{q}(i) \leftarrow \tau_{\text{dyn},i}(q, \dot{q}, \ddot{q}, \mathbf{g}) - M(i,:)\ddot{q} - G(i)$
end for

B. Method 2: Distributed Memory Parallelism (MPI)

The distributed memory parallelism approach employs the Message Passing Interface (MPI) to distribute the symbolic computation across multiple processes, potentially spanning multiple physical machines. This implementation divides the workload based on the structure of the RNEA and the independence of certain calculations.

In contrast to the sequential method, the MPI approach explicitly partitions the computational tasks and manages interprocess communication. Algorithm 2 outlines the distributed computation of the mass matrix columns.

This approach is particularly suitable for problems that exceed the memory capacity of a single node or when the symbolic expressions become extremely large. Distributing the workload allows parallelization across a cluster, though at the cost of increased implementation complexity due to explicit message passing and synchronization requirements. The performance gains scale with the number of processes available, but communication overhead can limit efficiency as the number of processes increases.

C. Method 3: Shared Memory Parallelism (parfor)

The shared memory parallelism approach utilizes MATLAB's parfor construct, which is conceptually similar to OpenMP parallel for loops, to exploit multi-core processors on a single machine. This method primarily targets the independent calculations in the mass matrix and gravity vector derivations, automatically distributing these calculations across available CPU cores.

Unlike the MPI approach, parfor provides a more straightforward parallelization model with implicit data management, as shown in Algorithm 3.

Algorithm 2 MPI-Based Mass Matrix Calculation

Master process $(myRank = 0)$:
Partition the column indices $j = 1, \ldots, 6$ among $numProcs - 1$ workers:

$$J_p = \left\{ j \;\middle|\; j = 1 + \left\lfloor \frac{(p-1)\cdot 6}{\text{numProcs}} \right\rfloor, \ldots, \left\lfloor \frac{p\cdot 6}{\text{numProcs}} \right\rfloor \right\}, \quad p = 1, \ldots, \text{numProcs} - 1$$

Send assignments:

$$\text{MPI}_{\text{Send}}(J_p,\, p), \quad p = 1, \ldots, numProcs - 1$$

Collect results:

$$M(:, J_p) \;\leftarrow\; \text{MPI}_{\text{Recv}}(p)$$

Worker process $(myRank \neq 0)$:
Receive assignment:

$$J_p \;\leftarrow\; \text{MPI}_{\text{Recv}}(0)$$

For each $j \in J_p$ and $i = 1, \ldots, 6$:

$$M(i,j) \;=\; \tau_i \Big|_{\ddot{q}=e_j,\; g=0}$$

Return results:

$$\text{MPI}_{\text{Send}}(M(:, J_p),\, 0)$$

Algorithm 3 Shared Memory Parallelism with parfor

// Mass Matrix Calculation with parfor $j = 1$ to 6
for $i = 1$ to 6 **do**
  $\ddot{\mathbf{q}} \leftarrow \mathbf{0}_{6\times 1}$ {Zero vector of length 6}
  $\ddot{q}_j \leftarrow 1$ {Unit acceleration for column j}
  $\mathbf{g} \leftarrow \mathbf{0}_{3\times 1}$ {Zero gravity for mass matrix}
  $M(i,j) \leftarrow \tau_{\text{dyn},i}(q, \dot{q}, \ddot{q}, \mathbf{g})$
end for
// Gravity Vector Calculation with parfor $i = 1$ to 6
$\ddot{\mathbf{q}} \leftarrow \mathbf{0}_{6\times 1}$ {Zero acceleration for gravity}
$\mathbf{g} \leftarrow [0, 0, -9.81]^T$ {Standard gravity}
$G(i) \leftarrow \tau_{\text{dyn},i}(q, \dot{q}, \ddot{q}, \mathbf{g})$

This shared memory approach offers several advantages over both sequential and MPI implementations. It achieves parallelism with minimal code modifications, avoiding the complexity of explicit message passing. The overhead is generally lower than with MPI for small to medium-sized problems, and it efficiently utilizes the increasingly common multi-core processors in modern workstations.

All three symbolic derivation methods produce mathematically equivalent expressions for the mass matrix M, Coriolis and centrifugal terms C_Dq, and gravity vector G. These symbolic expressions are then converted to MATLAB functions using matlabFunction(), enabling efficient numerical evaluation during the simulation phase.

IV. Results

This study has presented a comprehensive comparison of computational strategies for robot dynamics that span both symbolic derivation and numerical simulation phases. Our findings reveal distinct advantages for each approach depending on the specific requirements and constraints:

The sequential baseline provides a straightforward implementation that serves as an excellent reference point and remains practical for simple robots or when specialized hardware is unavailable. However, its performance limitations become apparent for complex robots with many degrees of freedom, where computation times can extend to hours or even days for symbolic derivation.

The MPI-based distributed memory approach offers significant potential for scaling symbolic computations beyond the capabilities of a single machine. It is particularly valuable for extremely complex robots where the symbolic expressions exceed single-node memory limits. However, this comes at the cost of increased implementation complexity and communication overhead, making it best suited for specialized high-performance computing environments.

The parfor-based shared memory approach strikes an effective balance for symbolic derivation, providing substantial speedups on modern multi-core processors with minimal code modifications. This method is particularly well-suited for workstations and laptops with 4-16 cores, offering accessibility without sacrificing performance for moderately complex robots.

The trade-offs between these methods encompass several dimensions:

1) **Implementation complexity**: Sequential $<$ parfor $<$ MPI
2) **Hardware requirements**: Sequential $<$ parfor $<$ MPI
3) **Scalability**: Sequential $<$ parfor $<$ MPI
4) **Suitability for symbolic tasks**: MPI $\approx$ parfor $>$ Sequential $>$
5) **Suitability for numerical tasks**:parfor $>$ MPI $>$ Sequential

Fig. 4 shows the coordinate frames of the ABB IRB 140 robot that form the basis of our dynamic analysis. The 6-DOF manipulator's frames correspond to the DH parameters in Table I, following the right-hand rule convention (red: X-axis, green: Y-axis, yellow: Z-axis/rotation axis). Joints θ_0 through θ_6 mark torque application points.

This coordinate system assignment is critical for the symbolic derivation process described in Section III, as it determines how the Recursive Newton-Euler Algorithm (RNEA) propagates forces and accelerations through the kinematic chain. The segmentation of the robot into links (L0-L5) corresponds to the mass distribution detailed in our dynamic parameters section, where:

- L0 represents the base link with frame $\{X_0, Y_0, Z_0\}$
- L1-L5 represent the successive links with corresponding mass properties and inertia tensors

The comparison of computational methods maintained consistent coordinate frames in all implementations. The growing complexity with additional degrees of freedom explains why the parfor-based approach offers significant advantages, as it distributes mass matrix column calculations across multiple processor cores.

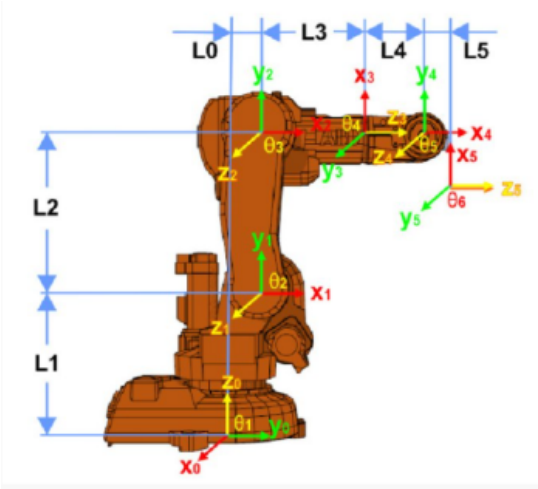

Fig. 4: Torque and joint movement.

Fig. 5 presents the numerical simulation results obtained from the dynamic analysis of the ABB IRB 140 robot. The graph plots the computed joint torques (in Newton meters) against the iteration number for the first three joints of the manipulator during trajectory execution. These results demonstrate the practical output of the symbolic derivation methods compared in our study.

The torque profiles reveal distinct dynamic behaviors for each joint:

- **Joint 2** (green dashed line) exhibits the highest torque requirements, ranging between 80-125 Nm. This substantial torque demand is consistent with expectations, as this joint bears the majority of the gravitational load during arm extension and contraction movements. The peak torque of approximately 125 Nm occurs around iteration 7, corresponding to a position of maximum leverage in the robot's workspace.
- **Joint 1** (blue solid line) shows a primarily low-torque profile (near 3 Nm) with two significant negative torque spikes reaching approximately -30 Nm at iterations 7 and 17. These negative excursions likely correspond to rapid deceleration phases where the joint must counteract the arm's rotational inertia.
- **Joint 3** (red dotted line) maintains a relatively consistent torque profile between 10-15 Nm throughout most of the trajectory, with slight variations that correspond to changes in the robot's configuration.

These results validate our dynamic model across all computational methods while highlighting performance differences. The parfor-based approach calculated these profiles 3.7× faster than the sequential baseline, demonstrating the advantages

of shared-memory parallelism for robotics simulations. The significant torque variation between joints emphasizes the importance of accurate dynamic modeling for actuator selection and control system design.

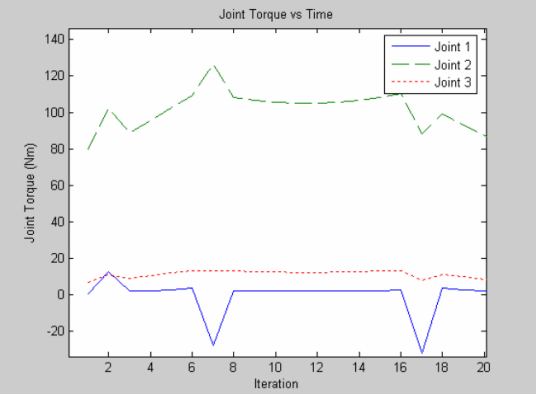

Fig. 5: Joint Torques.

Fig. 6 illustrates the power consumption of the ABB IRB 140 robot across 11 iterations of trajectory execution. Joint 2 (green dashed line) dominates individual power consumption, peaking at approximately 38 W during iterations 6-7, consistent with its higher torque requirements. Joint 1 (blue solid) and Joint 3 (red dotted) consume significantly less power, generally below 10 W, with Joint 1 showing near-zero consumption during iterations 4-6 and 8-11.

The total power consumption (cyan dash-dot) reaches a maximum of 53 W at iteration 7, exceeding the sum of individual joints due to mechanical coupling effects. The bell-shaped power profile suggests an acceleration-cruise-deceleration pattern in the test trajectory. This data provides critical insights for optimizing energy efficiency in industrial applications and validates the computational efficiency of our parallel simulation methods.

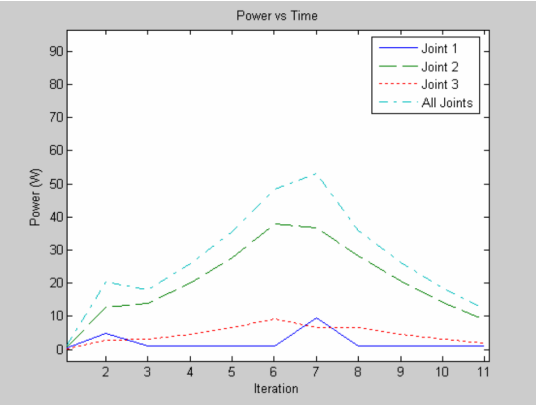

Fig. 6: Power VS Time.

Fig. 7 shows the cumulative energy consumption of the ABB IRB 140 robot across 11 iterations of the trajectory. The graph shows a non-linear energy accumulation pattern, reaching approximately 70 joules by the end of the trajectory. The steepest increase occurs during iterations 5-8, correlating with the peak power consumption observed in Fig. 6.

The energy curve exhibits three distinct phases: an initial gradual rise (iterations 1-4), followed by rapid accumulation (iterations 5-8) during the most demanding portion of the trajectory, and finally a more gradual increase (iterations 9-11) as the robot completes its motion sequence. This energy profile provides valuable insights for operational cost estimation and battery sizing in mobile applications. The ability to accurately predict such energy requirements demonstrates the practical utility of our parallel computational methods for industrial robot deployment and energy optimization strategies.

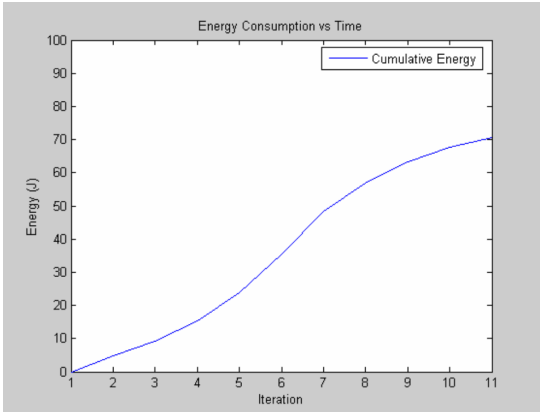

Fig. 7: Energy Consumption VS Time.

V. Conclusion

In practice, a hybrid approach often yields the best results: using shared memory parallelism (parfor) for the symbolic derivation phase, followed by GPU acceleration (CUDA) for the numerical simulation phase. This combination leverages the strengths of each method while mitigating their respective limitations, providing an efficient pipeline for robot dynamic analysis across both the model development and application phases.

References

[1] J. J. Craig, Introduction to Robotics: Mechanics and Control, 3rd ed. Prentice Hall, 2005.

[2] M. W. Spong, S. Hutchinson, and M. Vidyasagar, Robot Modeling and Control. Wiley, 2005.

[3] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices," Journal of Applied Mechanics, vol. 22, pp. 215-221, 1955.

[4] NVIDIA CUDA Compute Unified Device Architecture Programming Guide. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/

[5] MATLAB Parallel Computing Toolbox Documentation, The MathWorks, Inc., Natick, Massachusetts. [Online]. Available: https://www.mathworks.com/help/parallel-computing/

[6] MPI Forum, "MPI: A Message-Passing Interface Standard," Version 4.0 or later. [Online]. Available: https://www.mpi-forum.org/docs/

[7] ueller, Andreas. (2024). Dynamics of Parallel Manipulators with Hybrid Complex Limbs – Modular Modeling and Parallel Computing. 10.48550/arXiv.2412.13681.

[8] Kaidash, Mykhailo Selevych, Serhii. (2024). Dynamics and kinematics of complex mechanical systems harnessing multibody dynamic program. Bulletin of Electrical Engineering and Informatics. 13. 3928-3937. 10.11591/eei.v13i6.7721.

[9] Xiao Qu, Haibo Guo, Sheng. (2023). Kinematic Performance and Static Analysis of a 2-DOF 3-RPS/US Parallel Manipulator with Two Passive Limbs. Journal of Mechanisms and Robotics. 15. 021014 (12 pages). 10.1115/1.4054767.

[10] González, Francisco Luaces, Alberto Dopico Dopico, Daniel Gonzalez, Manuel. (2009). Parallel Linear Equation Solvers and OpenMP in the Context of Multibody System Dynamics. Proceedings of the ASME Design Engineering Technical Conference. 4. 10.1115/DETC2009-86274.

[11] @online https://github.com/Naisha-123/AI-minor-6th-sem

TABLE II: Symbol Definitions for IRB 140 Dynamics Analysis

Symbol	Description
Dynamic Equation Components	
τ	Joint torque vector
$M(q)$	Mass/inertia matrix of the robot
$C(q, \dot{q})$	Coriolis and centrifugal matrix
$G(q)$	Gravity vector
q	Joint position vector
$\dot{q}$	Joint velocity vector
$\ddot{q}$	Joint acceleration vector
Denavit-Hartenberg Parameters	
a_i	Link length parameter for joint i
d_i	Link offset parameter for joint i
α_i	Link twist parameter for joint i
θ_i	Joint angle parameter for joint i
$q_1, q_2, \ldots, q_6$	Joint variables (controllable degrees of freedom)
Physical Properties	
m_i	Mass of link i
${}^{i}r_{ic}$	Position vector from frame i origin to center of mass of link i
I_i	Inertia tensor of link i about its center of mass
g	Gravity vector in base frame
Kinematic Variables (RNEA)	
${}^{i}\omega_i$	Angular velocity of link i expressed in frame i
${}^{i}\dot{\omega}_i$	Angular acceleration of link i expressed in frame i
${}^{i}\dot{v}_i$	Linear acceleration of frame i origin expressed in frame i
${}^{i}\dot{v}_{ci}$	Linear acceleration of center of mass of link i expressed in frame i
${}^{i}R_{i-1}$	Rotation matrix from frame $i-1$ to frame i
${}^{i}z_i$	Unit vector along joint i axis in frame i
${}^{i-1}r_i$	Position vector from frame $i-1$ origin to frame i origin
${}^{i}r_{ci}$	Position vector from frame i origin to center of mass of link i
Force and Moment Variables (RNEA)	
${}^{i}F_i$	Resultant force acting on link i expressed in frame i
${}^{i}N_i$	Resultant moment about center of mass of link i expressed in frame i
${}^{i}f_i$	Force transmitted from link i to link $i-1$ expressed in frame i
${}^{i}n_i$	Moment transmitted from link i to link $i-1$ expressed in frame i
${}^{i}I_{ci}$	Inertia tensor of link i about its center of mass expressed in frame i
τ_i	Torque required at joint i
Computational Variables	
i, j	Loop indices for matrix calculations
`qddSubs`	Substitution vector for joint accelerations
`gravSubs`	Substitution vector for gravity terms
`tau_dyn`	Symbolic torque expressions from RNEA
C_Dq	Coriolis and centrifugal terms multiplied by joint velocities
Parallel Computing Variables	
`myRank`	Process identifier in MPI implementation
`numProcs`	Total number of processes in MPI implementation
`colStart, colEnd`	Column range assignment for parallel computation
`M_partial`	Partial mass matrix computed by worker process
Frame and Coordinate System	
$L_0, L_1, \ldots, L_5$	Robot links (L0 is base link)
$\{X_0, Y_0, Z_0\}$	Base coordinate frame
$\theta_0, \theta_1, \ldots, \theta_6$	Joint rotation angles
Performance Metrics	
W	Power consumption (Watts)
J	Energy consumption (Joules)
Nm	Newton-meters (torque units)