Journal of Robotics and Control (JRC)
Volume 5, Issue 1, 2024
ISSN: 2715-5072, DOI: 10.18196/jrc.v5i1.20582

Addressing Challenges in Dynamic Modeling of Stewart Platform using Reinforcement Learning-Based Control Approach

Hadi YADAVARI [1*],Vahid TAVAKOL AGHAEI [2], Serhat İKİZOĞLU [3]
[1] Department of Mechatronic Engineering, Istanbul Technical University, Istanbul, Turkiye
[2] AVL Research and Development Center, Istanbul, Turkiye
[3] Department of Control and Automation Engineering, Istanbul Technical University, Istanbul, Turkiye
Email: [1] yadavari@itu.edu.tr, [2] vahit.tavakol@avl.com
[3] ikizoglus@itu.edu.tr
*Corresponding Author

Abstract—**In this paper, we focus on enhancing the performance of the controller utilized in the Stewart platform by investigating the dynamics of the platform. Dynamic modeling is crucial for control and simulation, yet challenging for parallel robots like the Stewart platform due to closed-loop kinematics. We explore classical methods to solve its inverse dynamical model, but conventional approaches face difficulties, often resulting in simplified and inaccurate models. To overcome this limitation, we propose a novel approach by replacing the classical feedforward inverse dynamic block with a reinforcement learning (RL) agent, which, to our knowledge, has not been tried yet in the context of the Stewart platform control. Our proposed methodology utilizes a hybrid control topology that combines RL with existing classical control topologies and inverse kinematic modeling. We leverage three deep reinforcement learning (DRL) algorithms and two model-based RL algorithms to achieve improved control performance, highlighting the versatility of the proposed approach. By incorporating the learned feedforward control topology into the existing PID controller, we demonstrate enhancements in the overall control performance of the Stewart platform. Notably, our approach eliminates the need for explicit derivation and solving of the inverse dynamic model, overcoming the drawbacks associated with inaccurate and simplified models. Through several simulations and experiments, we validate the effectiveness of our reinforcement learning-based control approach for the dynamic modeling of the Stewart platform. The results highlight the potential of RL techniques in overcoming the challenges associated with dynamic modeling in parallel robot systems, promising improved control performance. This enhances accuracy and reduces the development time of control algorithms in real-world applications. Nonetheless, it requires a simulation step before practical implementations.**

Keywords*—*Stewart Platform; Dynamic Modelling; Reinforcement Learning; Deep Learning; Control.

I. Introduction

The primary goal of this study is to address the dynamical modeling challenges encountered in the control of the Stewart platform by utilizing a reinforcement learning (RL) approach. The Stewart platform, widely used in various applications such as flight simulators, driving simulators, and vibration testing for large structures, poses unique difficulties in its control due to its complex dynamics [1]–[3]. By employing an RL approach, the study aims to develop a solution that can effectively handle the complex dynamical modeling requirements of the Stewart platform. One of the goals in the field of artificial intelligence is to tackle complex problems by leveraging high-dimensional sensory data [4]. RL is a specialized branch of Machine Learning (ML) that revolves around an agent's interaction with its environment, guided by specific policies to maximize future rewards [5]. The agent's objective function is to optimize the cumulative sum of these rewards, with the Bellman equation serving as the foundation for defining optimal behavior. The agent's learning process is driven by a reward-penalty scheme, where the quality of selected actions from the policy space determines the outcomes [6]. In optimal control theory, a perfect system model with comprehensive descriptions is typically assumed [7]. However, such models often encounter issues such as modeling errors, uncertainties, and computationally expensive approximations. In contrast, RL operates directly on measured observations, encompassing uncertainties and non-linearities inherent in the system. Consequently, when dealing with complex systems and situations where classical analytical methods may yield inadequate control performance, RL is a favorable choice [8]–[12].

Within the literature, numerous neural network methodologies have been proposed to address the forward kinematics of the Stewart platform which is complex due to a set of high nonlinear equations [13]–[16]; but in terms of RL, there are few studies only. In a most recent study done by [17], deep RL algorithms to tune the gain parameters of a PID controller are used. This approach facilitated continuous learning and tuning of the controller's parameters. Different from this study, we show how to learn the forward dynamical control block using deep RL instead of only tuning PID control gains. In pursuit of this objective, we first delve into a comprehensive study of the platform's dynamics. If the dynamical model of the system was known precisely, we could perfectly control the platform. The kinematic model represents how the platform moves, but the dynamic model describes why the platform moves [18]. Control and simulation greatly rely on dynamic modeling, as it plays a key role in these domains. Unlike serial robots, the dynamic modeling of parallel robots is complicated due to the closed-loop kinematics inherent in their design [10], [19].

There have been various suggestions for conducting dynamic analysis on parallel manipulators. One commonly used approach is the traditional Newton-Euler formulation, which is also employed to analyze the dynamics of general parallel manipulators. This formulation offers a framework for assessing the forces and torques acting on the system, allowing for a comprehensive understanding of its dynamic behavior [20], [21]. To accurately describe the system dynamics within this formulation, it is crucial to derive the equations of motion for each leg and the moving platform. A preferable approach for achieving the dynamic formulation involves carefully selecting a set of independent generalized coordinates and subsequently deriving the dynamic equations using these coordinates and their corresponding time derivatives. Typically, these coordinates correspond to the positions of the moving platform. To achieve this type of formulation, it is necessary to eliminate internal forces and other passive joint variables. However, this process results in a large number of equations, which can have a negative impact on computational efficiency. We also have the Lagrangian formulation that proves to be highly effective in eliminating undesired reaction forces. However, the closed-loop structure of parallel manipulators imposes constraints that make it challenging and impractical to obtain explicit equations of motion using a group of separate generalized coordinates [22], [23].

As articulated, dynamical modeling has its fair share of challenges. Here, we demonstrate one method to enhance comprehension of the subject matter. We derive dynamical equations of motion using the principle of virtual work and the notion of link Jacobian matrices, as explained in [24]. Python libraries for symbolic mathematics (SymPy [25]) and numerical computing (NumPy [26]) are used to formulate and solve equations of motions represented in [24]. The primary challenge is to formulate the equation. Then we solve these equations by integrating forward in time. We show that deriving the dynamical model is a prohibitive task see Appendix A, and the final model is inaccurate and contains many simplifications; consequently, it is not suitable for real-time applications. While existing literature has made some attempts to address feedforward control methods using a reinforcement learning approach in some cases [27], [28], our current knowledge indicates a notable absence of such investigations in the domain of the Stewart platform. Therefore, we replace the classical feedforward inverse dynamic block with an RL agent to apply the required actions, which are the leg's forces here, for different trajectory states. We present an RL control topology to benefit from existing classical control topologies, inverse kinematic modeling, and the inverse dynamic of the system. We use the RL in a hybrid mode that helps to increase the performance of the control.

Due to the complexity of dynamic modeling, obtaining its derivation through conventional methods is challenging. Consequently, many opt to omit dynamic models in feedforward control, relying solely on feedback control in real-world applications. The proposed approach has the potential to offer a solution to overcome this challenge. RL also offers a dynamic approach that adapts to complex and nonlinear dynamics, mitigating the shortcomings of classical feedforward inverse dynamic blocks. By leveraging RL, the control system becomes more adept at learning optimal strategies, thereby enhancing precision while concurrently reducing the development time traditionally associated with precise control algorithms [29]. This explicit integration of RL directly tackles the identified challenges and presents a promising avenue for improving Stewart platform control in real-world applications.

We benefit from three Deep Reinforcement Learning (DRL) algorithms with two model-based RL algorithms. We first employ three DRL algorithms: the asynchronous advantage actor-critic (A3C) algorithm [30], the Deep Deterministic Policy Gradient (DDPG) approach [31], and the Proximal Policy Optimization (PPO) technique [32] to send force action to six legs motor directly beside the PID controller force output. Then we try two model-based RL algorithms, namely probabilistic inference for learning control (PILCO) [33] and model-based policy optimization (MBPO) [34], first to learn the dynamic model of the entire system and then utilize it to control the Stewart platform like feedforward control.

In a second attempt to improve the work carried out in [17], we propose a hybrid RL algorithm to learn a dynamical model of the system, resulting in more sample efficiency, well-suited to real-world applications like robotics [35], [36]. Even though model-free RL algorithms have succeeded in many areas, like video games and robotics, high sample complexity can limit the usage of model-free algorithms to simulated environments [31], [37]–[39]. Model-based RL algorithms use significantly fewer samples. Model-based methods extract more valuable information and are more data efficient than model-free algorithms. However, they suffer from model bias, meaning the model assumes it learned the environment's dynamic accurately. However, a poorly learned model may result in poor performance [34]. As we explain in Section IV the selected model-based RL algorithms address the model bias differently. Like three model-free algorithms, the two model-based RL algorithms are highly suitable to handle the continuous action-state spaces characteristic of the Stewart platform. PILCO relies on analytic gradient computation, and MPBO utilizes ensembles of models. By employing the probabilistic model (PILCO) and ensembles (MPBO), these two model-based algorithms are capable of achieving model-free performance with significantly fewer samples. To sum up, we experiment with five distinct RL algorithms using a proposed control framework. By adding the learned feedforward control topology by model-free and model-based RL algorithms to the existing PID controller, we demonstrate a noticeable enhancement in the overall control performance of the Stewart platform.

Regarding the search for the most effective strategy for the control of robot manipulators, DRL has demonstrated its efficiency. Nonetheless, there is considerable scope for enhancing its applicability in controlling parallel robots, motivating us to explore ways to fill this gap. The current paper's contributions are driven by the need for a dynamic model system that enhances the control performance of parallel robots while reducing the complexity of deriving the system's dynamic model. To summarize, these contributions, facilitated by the RL algorithm, can be outlined as follows:

1) **Enhancing the efficacy of a conventional control loop applied to parallel robots through the implementation of a suggested RL-based feed-forward loop.** The integration of a Predictive RL agent within the feedforward control loop, in conjunction with the classical control loop, augments the collective control performance.
2) **Simplifying the requirements for deriving the classical dynamic model used in the feed-forward control loop.** The suggested RL topology eliminates the necessity to formulate a complex dynamical model which is also a time-intensive task.
3) **Performing performance comparison with five DRL algorithms to explore the control capability of a parallel robot.** We employ a combination of model-free and model-based RL algorithms and subsequently conduct a comparative analysis, explaining the respective advantages and disadvantages within the context of the Stewart platform.

The rest of this paper is organized as follows: After reviewing kinematics and a common control strategy for the Stewart platform in Section II, we try to achieve a dynamic model of the platform via a classical method in Section III. In Section IV we present our approach to replacing the feed-forward dynamical model with a RL agent. The experimental setup and results are provided in Section V. The final observations of the paper and possible future works are presented in Section VI.

II. Kinematics and control strategy of Stewart platform

Calculation of the inverse kinematic of the Stewart platform is straightforward [40]. Therefore, we perform inverse kinematic modeling to improve the RL agent and final control performance. The explicit mathematical nature of inverse kinematics in terms of parallel robots accelerates the learning process for the RL agent. Because, by providing precomputed solutions, this approach substantially reduces the time and computational resources required for the RL agent to deduce these kinematic relationships independently. Fig. 1 illustrates a drawing of the kinematics and coordinate system of the Stewart platform.

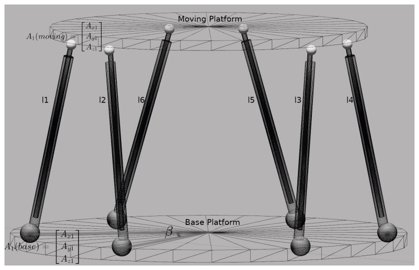

Moving Platform
$A_1(moving) = \begin{bmatrix}A_{x1}\\A_{y1}\\A_{z1}\end{bmatrix}$
l1
l2
l6
l5
l3
l4
Base Platform
β
$A_1(base) = \begin{bmatrix}A_{x1}\\A_{y1}\\A_{z1}\end{bmatrix}$

Fig. 1. Drawing of the kinematics and coordinate system of the Stewart platform [17]

There are two coordinate systems, base B_{xyz} and moving platform M_{xyz}. Since we have six legs in the Stewart platform design, we have six attachment points in both the base and motion platforms. In the inverse kinematic, the goal is to calculate the length of each leg $\{l_1, l_2, l_3, l_4, l_5, l_6\}$ given the pose of the moving platform, which means the position vector, $P = [P_x \; P_y \; P_z]$, and the orientation vector, $O = [\phi \; \theta \; \psi]$. We can calculate the connecting point coordinates through Equation (1).

$$A_i = \begin{bmatrix}A_{xi}\\A_{yi}\\A_{zi}\end{bmatrix} = \begin{bmatrix}r_p\cos(\upsilon_i)\\r_p\sin(\upsilon_i)\\z\end{bmatrix}, \begin{cases}\upsilon_i = \frac{i\pi}{3} - \frac{\beta}{2} & i = 1,3,5\\ \upsilon_i = \upsilon_{i-1} + \beta & i = 2,4,6\end{cases} \tag{1}$$

Therefore, we have all attachment points coordinates (B_i and M_i) for the given separation angles of the υ_{bi} in the base platform and the υ_{mi} for the motion platform. However, we calculated each attachment point of the moving platform in its coordinate system. Nonetheless, given the position and orientation of the moving platform, we want to calculate each point's coordination regarding the base. Coordinate transformations simplify mathematical expressions, decouple limb motions, and provide a unified representation. In order to transform the moving platform coordinate frame with respect to the base platform, we convert the pose of the end-effector using a translation position vector P Equation (2a) and a rotation matrix ${}^{B}R_{T}$ Equation (2b) [41].

$$P = [P_x \ P_y \ P_z]^T \tag{2a}$$

$$^{B}R_T = \begin{bmatrix} c(\theta)c(\psi) & c(\psi)s(\phi)s(\theta) - c(\phi)s(\psi) & s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta) \\ c(\theta)s(\psi) & c(\phi)c(\psi) + s(\phi)s(\theta)s(\psi) & c(\phi)s(\theta)s(\psi) - c(\psi)s(\phi) \\ -s(\theta) & c(\theta)s(\phi) & c(\phi)c(\theta) \end{bmatrix} \tag{2b}$$

Equation (3) calculates the position vector using the position and orientation matrices. The actuator length is then obtained as $l_i = ||L_i||$.

$$L_i = (^{B}R_T)M_i + P - B_i \quad i = 1, 2, ..., 6. \tag{3}$$

where M_i and B_i are the attachment point coordinates in moving and base platform frames, respectively.

After inverse kinematic calculation, we want to control the platform. It means we want the moving platform to reach the desired reference pose. Plenty of control techniques have been developed for robotic manipulators, such as a nonlinear model and a multi-input/multi-output for multiple degrees of freedom robots [42]–[45]. However, in industry, controllers usually control individual joints through drivers linearly [46]. Fig. 2 shows one of the known control topologies of the Stewart platform, which uses the length measurement of each leg as feedback in the closed-loop system [47].

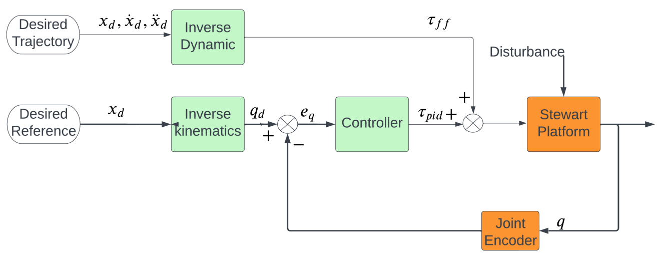


Fig. 2. General joint space control topology of the Stewart platform with Feedforward control extension.

In Figure 2, the parameter q is the linear displacement of an actuated prismatic joint which is equal to each leg's length and q_d is the desired linear displacement of legs which is calculated by the inverse kinematic model. Instead of the Task Space control, measuring the position and orientation of the end-effector, the controller is implemented in the joint space, where we calculate the joint space error e_q. This control topology converts the desired trajectory to the desired actuators' lengths through the inverse kinematic model, which we derived mathematically. Then, the controller calculates the required actuator torque τ. Having the inverse dynamic model of the system has the potential to enhance the efficiency of the controller, as illustrated in the control topology. Nonetheless, in Section III we demonstrate that obtaining and resolving the dynamic model of the Stewart platform is intricate. To address this complication, as outlined in Section IV, we substitute this loop with an RL agent. This alteration is intended to address the aforementioned challenge.

III. Deriving and Solving Dynamical Equations of Motion

To derive the dynamical equations of the motion, the principle of virtual work and the notion of link Jacobian matrices, as outlined in [24], are employed and derived in Appendix A. As demonstrated in the appendix, the main challenge is to formulate the equations. To address this, we employ Sympy to methodically compose and derive these equations in a concise and organized manner. The code is accessible in the open-source repository provided by [48]. Despite leveraging the Sympy library and its advantages for handling the formulation, it remains intricate in that context. The task of deriving the inverse kinematics of the Stewart platform with considerations like the elimination of joint frictions is notably laborious. Once we engage in experiments involving the simulated Stewart platform in Gazebo [49], it becomes evident that its dynamic model is considerably more detailed and intricate than the one we formulated mathematically. It contains features like friction, damping, and other dynamical values. We ignored most of these features to model the Stewart platform's inverse dynamic. Otherwise, formulation and solving dynamic equations would be much more challenging tasks.

Upon achieving this derived formulation, the next required step involves real-time solutions to enable its application.

A. Solving Dynamical Equations of Motion

We solve the dynamical equations using numerical methods. For solving the equations, we define a trajectory for the moving platform. It means that we need to specify X , $\dot{X}$, and $\ddot{X}$. The algorithm used to solve the inverse dynamic for specific points is described in Algorithm 1:

Algorithm 1: Numerical calculation of the inverse dynamic for the given point

```
    **Calculate the Jacobians:** $Jp$, $Jx$, and $Jy$ are calculated for the given $X$ and inertial properties;
    **Calculate the Forces:** $F_p$, $F_x$, $F_y$, and $F_z$ are calculated for the given $X$, $\dot{X}$, $\ddot{X}$, and inertial properties;
    **Calculate the Leg Forces:** The required leg forces are calculated via inverse dynamic Equation 45.
```

We use the Solver module of the SymPy library to numerically solve the derived inverse dynamics. For every value of X, $\dot{X}$, and $\ddot{X}$ without considering orientation, it takes about 6 to 7 seconds to complete all these calculations. In the case of orientation, the calculation takes too long. Some test point results are given in Table I. The experiments were accomplished on Amazon EC2 (Amazon Elastic Compute Cloud) R6a instances [50]. These instances are equipped with 3rd generation AMD EPYC processors. Specifically, we utilized the r6a.xlarge instance type, which includes 4 CPUs with a turbo frequency of 3.6 GHz and 32 GiB of memory.

Some points for orientation took too long (26 seconds and more) to calculate the final answer. To speed up the calculation of forces for a set of points (trajectory), Sympy expressions are converted to Numpy utilizing the Lambdify method. Inverse dynamic calculations take a long time and are complex to track. For this reason, we found it appropriate to make evaluations on various examples. During these processes, we observed a large jump in required forces for values of P_z around zero, which is not realistic to meet in practice. An example of this phenomenon is Example 1 where the trajectory is defined as in Equation (4). Related curves are presented in Fig. 3 (a), (b), (c).

$$X = [0.001 * \sin t, 0.001 * \sin t, 1 + \sin t, 0, 0, 0] \quad (4)$$

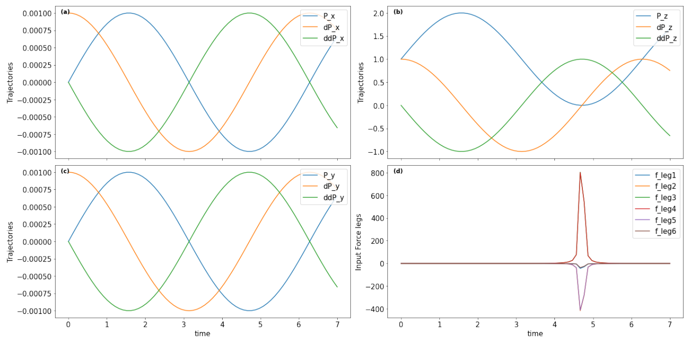


Fig. 3. Platform trajectories of Example 1 and the inverse dynamic solutions.

Fig. 3 (d) shows the inverse dynamic solution for the trajectory 4, which takes 9 min to solve for 70 separate time points between 0 to 7 seconds. Around 4.8 s we have P_z close to zero. In order to verify our observation about the trajectory values around zero, we performed further experiments. First, by moving the P_z trajectory a bit up as shown in Equation (5) and Fig. 4 (a), (b), (c), we observe that the effect of this point has been reduced but still causing a significant jump in the required forces. Fig. 4 (d) shows the inverse dynamic solution for this Example 2 trajectory, which took 8 min to solve for 70 separate time points between 0 to 7 seconds.

$$X = [0.001 * \sin t, 0.001 * \sin t, 1.1 + \sin t, 0, 0, 0] \quad (5)$$

Second, we move the P_z trajectory a bit down from 1.1 to 1 and increase the amplitude by 10 percent as shown in Equation (6) and Fig. 5 (a), (b), (c). Although not as high as in Example 1, we observe still large values for the forces corresponding

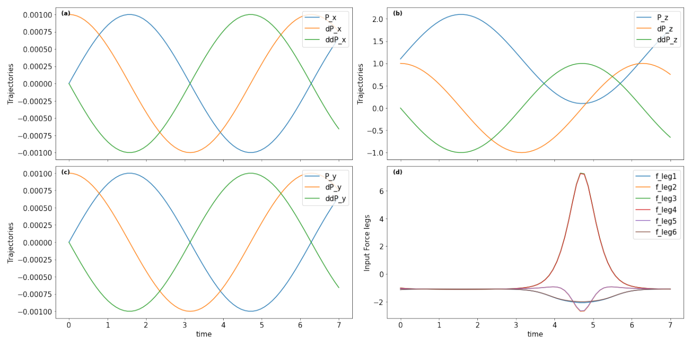


Fig. 4. Platform trajectories of Example 2 and the inverse dynamic solutions.

to the near-zero values of P_z. The inverse dynamic solution is shown in Fig. 5 (d).

$$X = [0.001 * \sin t, 0.001 * \sin t, 1 + 1.1 * \sin t, 0, 0, 0] \quad (6)$$

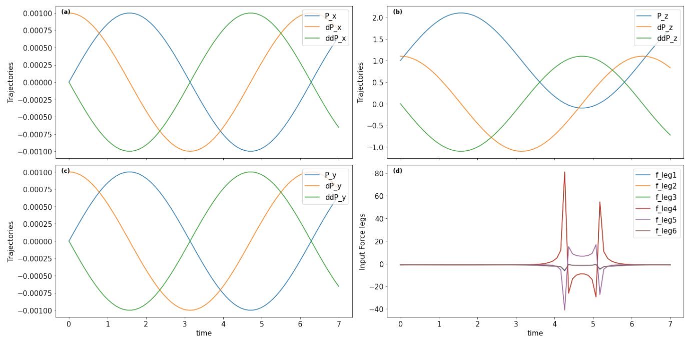


Fig. 5. Platform trajectories of Example 3 and the inverse dynamic solutions.

Finally, we investigate the fourth example limiting z as defined in Equation (7) and shown in Fig. 6 (a), (b), (c) as Example 4. Fig. 6 (d) shows the inverse dynamic solution for this trajectory, which takes 9 min to solve for 70 separate time points.

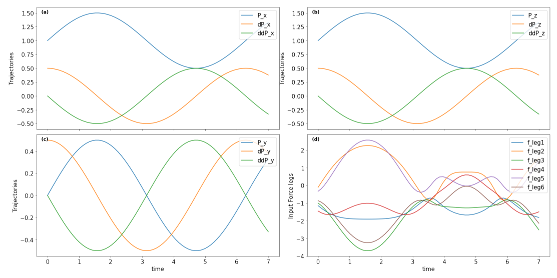


Fig. 6. Platform trajectories of Example 4 and the inverse dynamic solutions

$$X = [1 + 0.5 * \sin t, 0.5 * \sin t, 1 + 0.5 * \sin t, 0, 0, 0] \quad (7)$$

TABLE I. SIMPLE POINTS CALCULATION OF THE INVERSE DYNAMIC FORMULATION

Test Point	X	$\dot{X}$	$\ddot{X}$	Result leg Forces	run-time(s)
point 1	[1,1,1,0,0,0]	[1,0,0,0,0,0]	[0,0,0,0,0,0]	[-0.25,0.6,-2.5,-0.03,0.68,-2.3]	6.5
point 2	[2,1,1,0,0,0]	[1,1,1,0,0,0]	[0,0,0,0,0,0]	[-2.5,7,-9,0.35,8.9,-7.6]	6.5
point 3	[2,2,2,0,0,0]	[1,1,1,0,0,0]	[1,1,1,0,0,0]	[2.65,2.043,-11.79,7.09,5.53,-9.49]	6.6
point 4	[1,1,1,1.047,0,0]	[0,0,0,0,0,0]	[0,0,0,0,0,0]	[-1.4,0.36,-1.37, -1.9,-1.86, 1.2]	45.2
point 5	[1,1,1,1.047,1,0]	[0,0,0,1,1,1]	[0,0,0,0,0,0]	[6.6,-6.03,1.6,-1.69,1.15, 0.49]	112.1

This is the problem with this prolonged method for real-time control applications. We note that we even ignored dynamic features like friction and damping to simplify the formulation and calculations. In addition, we assumed that the manipulator Jacobian matrix Equation (28) is not singular. If J_p is singular, we can't solve the inverse dynamic equations. Even though using the inverse dynamic model in feedforward could increase the controller performance, the numerical model is too slow to respond. One could linearize around the equilibrium point to have a more straightforward and fast-to-run model. However, we aim to learn such a model through RL without explicitly deriving all formulation that ignores many dynamical features. The classical methods also solve this simplified model with further assumptions under linearization, which is not optimal. We aim to learn the complex dynamic model with all features through RL and utilize it in the feedforward control loop.

IV. FEED FORWARD CONTROL VIA REINFORCEMENT LEARNING

The disadvantages of the method which uses the inverse dynamic model and a classical feedback control loop structure as presented in Fig. 2 are the complexity of both derivation and solving of the equations. Determining the inverse dynamics is a tough task. In addition, following various simplifications, it becomes necessary to once again linearize this model around a designated equilibrium point for practical applications [51], [52]. Yet, this process is task-specific, and we need to do all steps in case new changes are made to the design and structure of the platform. In this section, we present an RL control topology to benefit from existing classical control topologies, inverse kinematic modeling. We use RL in a hybrid mode that helps to increase the performance of the control, as presented in Fig. 7.

We form RL and control blocks similar to the RL setup experimented in [17]. However, we change the action space completely from changing PID gains to applying force to each leg. As shown in Fig. 7, we also benefit from inverse kinematic modeling, feedback control loop, and PID controller. The essential components of the RL setting are as below:

1. Action space: The action space at time step t can be expressed as:

$$A_t = [f_1 \quad f_2 \quad f_3 \quad f_4 \quad f_5 \quad f_6] \tag{8}$$

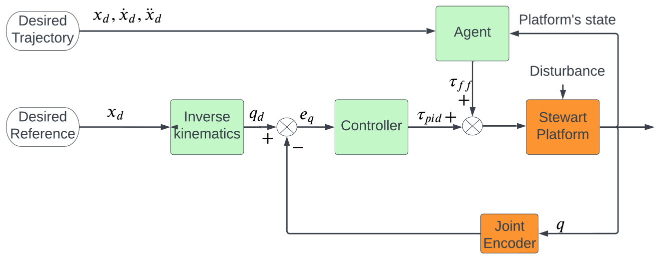


Fig. 7. Propsed RL topology in the control of Stewart platform in a feedforward manner

where f_1, f_2,..., and f_6 represent the applied force to each leg of the Stewart platform.

2. State space: We define our problem states as Equation (9):

$$X = [x \ y \ z \ \phi \ \theta \ \psi] \tag{9}$$

where X is the pose vector of the end-effector at time step t. It is worth noting that the pose velocity $\dot{X}$ and the difference between the target pose of the end-effector and the actual pose ΔX can be considered as additional state variables. However, it is important to be mindful that increasing the dimensionality of the state space can lead to slower convergence of the algorithms [53], [54]. However, we utilize ΔX directly in the reward setup, and the pose velocity $\dot{X}$ checks whether the episode is done.

For the three model-free DRL algorithms, we define a similar reward function introduced in [17]. We consider the well-known *reaching task* as our goal.

3. Reward setup: We choose a quadratic reward function similar to work in [55] and given in Equation (10), with a modification that in case of any goal reaching failure or instability of the system, the agent is being penalized significantly by a value of -1000.

$$R_t = \begin{cases} -\Delta X^T V \Delta X & \text{for } D < \delta \\ -1000 & \text{otherwise} \end{cases} \tag{10}$$

In Equation (10), $V \in \mathbb{R}^{4x4}$ is a diagonal positive definite weight matrix for the errors, $D = e_{xyz}$ is the distance to the goal value, and δ defined as a threshold distance to handle the falling of the platform. For the reward function defined in Equation (10) with the desired pose having index d, error vector is calculated as $\Delta X = [e_{xyd},\ e_\phi,\ e_\theta,\ e_\psi]$ containing all the error terms given in Equations (11a), (11b), (11c), and (11d).

$$e_{xyz} = \sqrt{(x - x_d)^2 + (y - y_d)^2 + (z - z_d)^2} \tag{11a}$$

$$e_{\phi} = |\phi - \phi_d| \tag{11b}$$

$$e_{\theta} = |\theta - \theta_d| \tag{11c}$$

$$e_{\psi} = |\psi - \psi_d| \tag{11d}$$

The three DRL algorithms utilize a parameterized policy function to enable the utilization of continuous actions or the ongoing adjustment of leg force values. This policy function takes the robot's state, represented as X, as inputs and generates continuous leg forces as outputs using its corresponding neural network. Nevertheless, directly learning the policy network comes with inherent high variability. This challenge prompted the introduction of actor-critic methods [56]–[58]. In this paradigm, the "Critic" assesses the value function $Q(s, a|\mathcal{W}_Q)$ using the Bellman equation, akin to Q-learning. Subsequently, the "Actor," parameterized by the function $\mu(s|\mathcal{W}_\mu)$", adjusts the policy distribution following the guidance provided by the Critic. While these three DRL algorithms share the foundational Actor-Critic framework, they diverge in their architectures, which we elucidate below.

The first algorithm we consider is the DDPG approach, an adaptation of the deterministic policy gradient (DPG) algorithm [59] that incorporates deep learning principles. DDPG stands as an off-policy and model-free Deep DRL algorithm capable of acquiring proficient policies for various tasks, even when presented with low-dimensional observations like joint angles and Cartesian coordinates [31]. Implementing DDPG involves a relatively simple actor-critic architecture, employing parameterized actor and critic functions: $\mu(s|\mathcal{W}_\mu)$ and $Q(s, a|\mathcal{W}_Q)$. To enhance learning stability, DDPG incorporates target networks for both the actor and critic functions. This entails creating duplicates of the actor and critic networks, referred to as $\mu'(s|\mathcal{W}_{\mu'})$ and $Q'(s, a|\mathcal{W}_{Q'})$, respectively. The next algorithm is A3C, an on-policy and model-free DRL approach. A notable benefit of A3C lies in its utilization of parallel actor-learners, which contribute to stabilizing the training process. A3C has proven successful in addressing a diverse range of continuous motor control challenges [30]. While DDPG is trained off-policy using samples from a replay buffer to mitigate sample correlations, A3C employs an alternative approach. Instead of relying on experience replay, A3C simultaneously runs multiple agents asynchronously across multiple instances of the environment. This strategy aims to reduce the issue of data correlation. For our third DRL algorithm, we utilize PPO, a data-efficient and dependable variant derived from the trust region policy optimization (TRPO) approach [60]. Similar to the prior algorithms, PPO is also a model-free, on-policy method that follows a pattern of gathering data from the policy and then undergoing multiple optimization epochs to enhance the policies.

In the subsequent algorithms, we explore two model-based RL approaches. However, the fundamental question arises: why should we consider employing a model-based RL algorithm? In general, an agent in RL can make decisions in two main ways, model-based and model-free. In model-based RL, the agent utilizes its model to decide what action to take. However, in model-free RL, without having a model, the agent tries to learn the optimum policy. The main question is if we need such a model to take action. Model-free RL algorithms have succeeded in many areas, like video games and robotics, in recent years [31], [37], [38]. However, high sample complexity limits their usage to simulated environments mostly. On the other hand, model-based RL algorithms use significantly fewer samples than model-free ones. Therefore, by learning a dynamical system model, we expect sample efficiency, which is very important in real-world applications like robotics [36]. In general, model-based methods extract more valuable information and are more data efficient than model-free algorithms. However, they suffer from model bias, meaning the agent thinks it accurately learned the environment's dynamic. Whereas, a poorly learned model results in poor performance. Many model-based RL algorithms have tried to address the model bias differently. Many successful machine learning applications are based on data augmentation [61]–[63]. Sutton in [64] presents a model-based Dyna algorithm in which a model is learned in a supervised manner through collected data and new data generated under the model. The policy improvement utilizes the model data. But, as a problem of model-based RL, modeling errors could cause diverging temporal-difference updates. In Dyna and standard RL framework, we want to maximize the expected return from acting according to policy π in the environment under some dynamics p:

$$\pi^\star = argmax_\pi \eta[\pi] = argmax_\pi \, \mathbb{E}_\pi \sum_{t=0}^{T} [\gamma^\infty r(s_t, a_t)] \tag{12}$$

However, the learned model is commonly reliable for a single-step predictive model. However, the modeling errors could sum up for the long horizon, resulting in poor performance for long rollouts.

We experiment model-based RL approach first with the MBPO algorithm which utilizes short rollouts from the predictive model rather than full-length rollouts starting the initial state distribution to gather data and update the policy [34]. MBPO addresses three key aspects to enhance the Algorithm: the parameterization of the predictive model p_θ, the policy optimization π based on model samples, and the method of querying the model for samples. In MBPO, the predictive model utilizes a bootstrap ensemble of dynamic models, denoted as $p_\theta^1, ..., p_\theta^B$, which are probabilistic neural networks that generate Gaussian parameterizations [34]. To ensure diversity in the dynamics models, MBPO uniformly selects one predictive model at random from the ensemble. This approach allows for the sampling of different dynamic models. For policy optimization, MBPO employs the soft actor-critic (SAC) algorithm [37]. When the horizon span k is short, MBPO uses the predictive model to conduct multiple short rollouts. This approach helps generate a substantial collection of model samples for policy optimization. This comprehensive set of model samples enables MBPO to take multiple policy gradient phases per environment sample, exceeding the capabilities of model-free algorithms [34].

As a second model-based RL approach we experiment with PILCO that uses its observed samples efficiently. The problem with the model-based methods is model errors. In PILCO, the dynamic model of the system is approximated by Gaussian processes. In this manner, PILCO addresses the model-bias problem of the model-based RL while using less sample of data. The assumption in PILCO or model-based RL is that we do not have prior or expert knowledge about the model of the system, like differential equations for the dynamics. However, we want to learn the model from scratch.

PILCO utilizes non-parametric probabilistic Gaussian processes (GPs) as the basis for its dynamic model, considering model uncertainties as noise in the system. To account for these uncertainties during planning and policy evaluation, PILCO incorporates them into its framework. For policy search and update, PILCO employs an analytic policy gradient method, enabling effective optimization of the policy [65]. PILCO considers model dynamic as Equation (13):

$$x_{t+1} = f(x_t, u_t) \tag{13}$$

The system under consideration involves continuous-valued states denoted by $x \in \mathbb{R}^D$ and controls denoted by $u \in \mathbb{R}^F$. The transition dynamics of this system, represented by the function f, are not known prior. The objective of policy improvement is to discover a deterministic policy or controller denoted as π, which maps states x to actions u such that $\pi(x, \theta) = u$. The goal is to minimize the expected return associated with the policy:

$$J^{\pi}(\theta) = \sum_{t=0}^{T} \mathbb{E}_{x_t}[c(x_t)] \tag{14}$$

where $c(x_t)$ is the negative reward or cost of being in state x at time t.

Regarding policy optimization, PILCO aims to maximize the expected cumulative reward within a finite time horizon, utilizing the learned Gaussian process (GP) model for each potential policy. This involves simulating the system forward in time and calculating the expected cumulative reward. The optimization problem is subsequently solved using gradient-based techniques such as the conjugate gradient or L-BFGS-B method. After optimization, PILCO continues collecting the dataset by executing the policy and updating the probabilistic model of the system dynamics. This allows the algorithm to learn effectively with relatively little data, which is particularly useful when data collection is expensive or time-consuming. However, we note that PILCO is very computationally expensive in model and policy optimization steps. In terms of the exploration-exploitation trade-off, PILCO employs a saturating cost function that facilitates natural exploration when the predictions are distant from the target [65]. This means that the policy explores more actively in situations where predictions are far from the target. Conversely, when predictions are close to the target, the policy remains close to the learned trajectory and focuses on exploitation. The fast learning speed of PILCO makes it suitable for controlling real-world applications such as robotics. However, it is worth noting that PILCO is currently limited to episodic setups.

V. Experiments Setup and Results

We experiment on the similar simulated Stewart platform presented in [17] with our newly proposed algorithm, where the same inertial properties of the experimented platform are shown in Table II. The inertial characteristics impact how the platform responds to external forces and influences the overall behavior that the learning process must adapt to.

TABLE II. Inertial Properties of the Experimented Stewart Platform

Link	Type	Mass (kg)	$I_{xx}(kg.m^2)$	$I_{yy}(kg.m^2)$	$I_{zz}(kg.m^2)$
platform	cylinder	0.1	0.065	0.065	0.128
bottom ball	sphere	0.01	0.00004	0.00004	0.00004
top ball	sphere	0.01	0.00001	0.00001	0.00001
cylinder	cylinder	0.1	0.02777	0.02777	0.00012
shaft	cylinder	0.1	0.027725	0.027725	0.00003

Our objective is to guide the moving platform to achieve the target pose (reaching task) defined as $[x = 0, y = 0, z = 1.1, \phi = 0, \theta = 0, \psi = 30]$, originating from the initial state characterized by the pose $[x = 0, y = 0, z = 0.2, \phi = 0, \theta = 0, \psi = 0]$. This transition entails a heave motion of $90\ \ \mathrm{cm}$ and a yaw rotation of $30\,\mathrm{deg}$ for the platform. It's worth noting that these specific poses were experimented with in a prior study [17], and we opted for the same target to facilitate comparative analysis and validate the effectiveness of the proposed methodology. Across all conducted experiments, we establish the episode count at 500, with a maximum of 200 steps allowed per episode. These values were determined based on a careful consideration of the learning process and computational efficiency. The choice of 500 episodes allows for a sufficiently iterative learning process, allowing the reinforcement learning agent to adapt and refine its strategies over a substantial number of training iterations. Meanwhile, setting a maximum of 200 steps per episode is a balance between capturing complex learning scenarios and managing computational resources effectively. At the beginning of each episode, the robot is positioned in the initial pose, while the agent's distance threshold is defined as $\delta = 1.5$ m. Within each learning episode, the platform is subject to a penalty if it deviates significantly from the desired final pose (surpassing a distance of 1.5 m). The penalty value is configured as -1000, as indicated in Equation 10. The penalty value was chosen to strongly discourage undesired actions and deviations during the learning process and the distance threshold was determined in alignment with a comparable setup to [17]. The weight matrix for error diagonals is established as $V = \text{diag}[1, 1, 1, 1]$. The neural network structure for all three algorithms, along with their respective hyperparameters, is detailed in Table III and Table IV, respectively. In the case of MBPO, the network architectures, and hyperparameters are specifically presented in Table V.

TABLE III. ARCHITECTURE OF THE NEURAL NETWORKS OF THE DRL ALGORITHMS

DRL algorithms	layer	Actor/target-Actor	Critic/target-Critic
PPO	input numbers	6	6
	1st activation	Relu.	Relu
	layer 1 units	400	400
	2nd activation	Relu	Relu
	layer 2 units	300	300
	3rd activation	-	Relu
	layer 3 units	-	200
	final activation.	sigmoid	linear
	final output nums.	3 (denormalized)	1
	std activation.	softplus	-
	std output nums.	6	-
A3C	input numbers	6	6
	1st activation	Relu.	Relu
	layer 1 units	400	400
	2nd activation	Relu	Relu
	layer 2 units	300	300
	3rd activation	-	Relu
	layer 3 units	-	200
	final activation.	sigmoid	linear
	final output nums.	6 (denormalized)	1
	std activation.	softplus	-
	std output nums.	6	-
DDPG	input numbers	6	6+3
	1st activation	Relu.	Relu
	layer 1 units	400	400+32
	2nd activation	Relu	Relu
	layer 2 units	300	300 + 16
	output activation.	sigmoid	linear
	output nums.	6 (denormalized)	1

For PILCO, a similar reward function configuration is used. In PILCO, the cost function imposes a penalty based on the Euclidean distance between the current state and the target state. Although a defined reward is obtained from the reaching task, it is not utilized in the policy optimization of PILCO. Instead, only distance penalties are employed to address the task, as specified in the PILCO paper [33]. In the PILCO setup, reaching the target with high speed often directs to overshooting, resulting in elevated long-term costs. Therefore, the effects of the failing task resulting in a negative -1000 reward are not directly considered in the PILCO's policy optimization, but it shows its effect as the Euclidean distance from the target is too high in case of failure. We determine the specific hyperparameters used in PILCO according to our reaching task environment shown in Table VI.

TABLE IV. ARCHITECTURE OF THE NEURAL NETWORKS OF THE DRL ALGORITHMS

DRL Algorithms	Hyper-parameter	Value
PPO	discount factor (γ)	0.99
	actor learning rate	0.0005
	critic learning rate	0.001
	update interval	5
	clip ration	0.1
	GAE parameter (λ)	0.95
	Entropy coefficient c2	0.01
A3C	discount factor (γ)	0.99
	actor learning rate	0.0005
	critic learning rate	0.001
	update interval	5
	Entropy coefficient (β)	0.01
DDPG	discount factor (γ)	0.99
	actor learning rate	0.0001
	critic learning rate	0.001
	batch size	64
	target network τ	0.001
	Ornstein-Uhlenbeck process - θ	0.15
	Ornstein-Uhlenbeck process - σ	0.2

TABLE V. HYPER-PARAMETERS AND NEURAL NETWORK ARCHITECTURE USED IN THE TRAINING OF MBPO

	Layer/HP	Value/Actor	Critic
SAC in MBPO	number of inputs	6+6	6+6
	1st activation	ReLU.	ReLU
	1st layer units	256	256
	output activation	linear	linear
MBPO	discount factor (γ)	0.99	
	learning rate	0.0003	
	target smoothing coe.(τ)	0.005	
	Temperature parameter (α)	0.02	

TABLE VI. HYPER-PARAMETERS OF PILCO

	Hyperparameter	value
GP (for each 6 GP model)	Length Scale	different for each GP
	Kernel Variance	different for each GP
	Noise Variance	different for each GP
Policy Search Hyperparameters	Number of Policy Updates	200
	Step Size	200
Exploration Hyperparameters	Exploration Noise	0.5
	Exploration Horizon	20
	Number of Rollouts	200

Fig. 8 shows the final reaching task rewards of the five RL algorithms, 3 DRL, and 2 model-based RL algorithms. We experiment with each algorithm 5 times and average over the runs to have a valid scientific comparison. As shown in Fig. 8, all five rewards started from negatively large values toward converged to zero.

The best steady performance is for PILCO with the minimum convergence step. The worst convergence is for MBPO even though it is more stable in the last steps in comparison to the other three model-free algorithms. In general, model-based

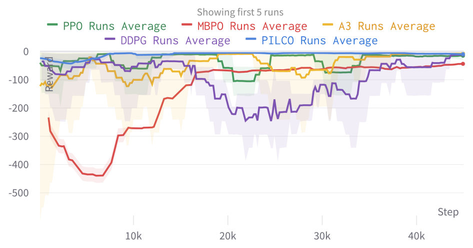
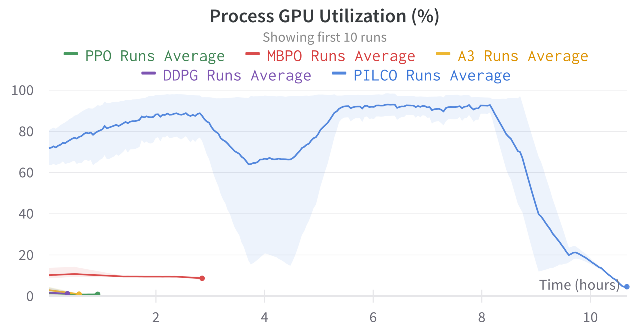


Fig. 8. Reaching task rewards of the five RL algorithms.

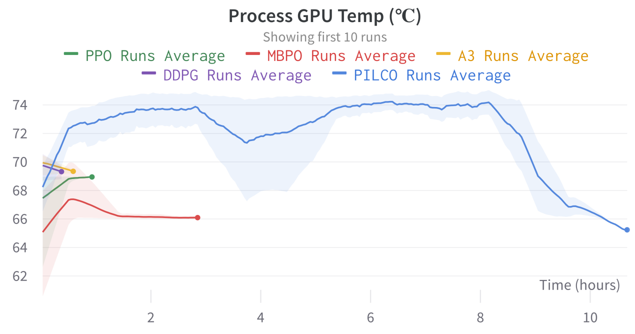
algorithms tend to exhibit more stable convergence compared to model-free algorithms.

The overall control performance of the three model-free algorithms is more stable than the result experimented in [17] with faster convergence. Comparing the quantitative time-domain response of the results obtained from the newly proposed method with those in [17], we observe an approximate 96% enhancement in rise time and a 75% improvement in settling time, despite a similar overshoot observed in [17] in the PILCO case. This underscores a substantial improvement achieved by integrating reinforcement learning (RL) into the new proposed structure, particularly in the feedforward loop section. It is noteworthy that the RL methodology applied in [17] focused solely on tuning the PID values of the controller, whereas in our approach, we learn a dynamic model. The utilization of this acquired knowledge contributes to an overall performance improvement alongside the classical PID controller. PILCO shows great performance, in terms of stability and convergence, against the other three model-free and one model-based RL algorithms. It learns the system's dynamics well and utilizes it to optimize the policy, resulting in the best performance PILCO often integrates a probabilistic model and incorporates uncertainty into its predictions. This can enhance its adaptability to varying conditions, contributing to stability in learning tasks. Furthermore, PILCO may leverage a model-based approach that refines its understanding of the system dynamics, leading to more efficient convergence The only disadvantage of the PILCO algorithm is that it is very computationally expensive. We show the result of GPU utilization in Fig. 9. As we see each training PILCO run takes over 10 hours, almost 10 times longer than model frees' training time, and 3 times longer than the other model-based algorithm (MBPO). In addition to the duration, GPU utilization and GPU temperature in training with the PILCO algorithm are very high as shown In Fig. 9. Overall, even though model-based algorithms are sample efficient, they are computationally expensive, requiring more computational resources.

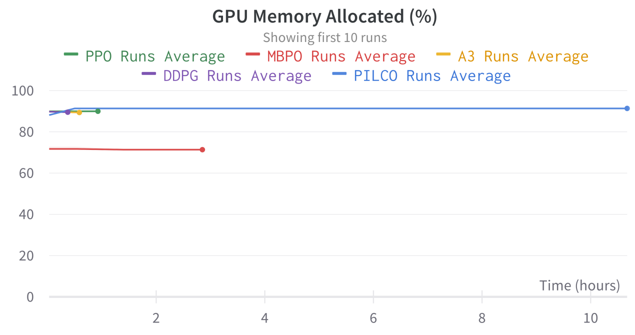
Regarding the time response performances of the five RL algorithms, we run an episode for each learned algorithm and compare their performances. Fig. 10 illustrates the time-domain performance of the moving platform's states for the



(a) Process GPU Utilization Percentage



(b) Process GPU Temperature



(c) GPU Memory Allocation Percentage

Fig. 9. Computation time and GPU usage of different RL algorithms in training

five RL algorithms. Our primary objectives for the reaching task involve achieving a heave of 1.1 with a 30-degree yaw. As depicted in Figure 10, PILCO exhibits the best performance, demonstrating minimum rise time with a stable response and the least steady-state error, particularly for these two main goals. The platform's roll, pitch, surge, and sway performance for the learned control scenario is also presented, showing comparable performance to heave and yaw in these four movements, surpassing PILCO in these aspects. It's worth noting that since we have minimal movement in roll, pitch, surge, and sway, we have magnified the y-axis for these variables to facilitate a clear examination of their time response. PILCO's probabilistic and model-based approach allows it to effectively capture system uncertainties, enabling better adaptability to dynamic changes in the environment. This adaptability contributes to PILCO's reduced rise time, as it can swiftly adjust its control policies in response to evolving conditions. Moreover, the incorporation of uncertainty modeling aids PILCO in minimizing steady-state errors. By accounting for and mitigating uncertainties, PILCO exhibits enhanced precision in maintaining the desired pose. The stability exhibited by PILCO can be linked to its comprehensive understanding of system dynamics through a probabilistic and model-based approach. This enables PILCO to navigate the learning process with increased robustness, resulting in more stable and reliable control performance. Table VII shows the time domain performances of the 5 algorithms.

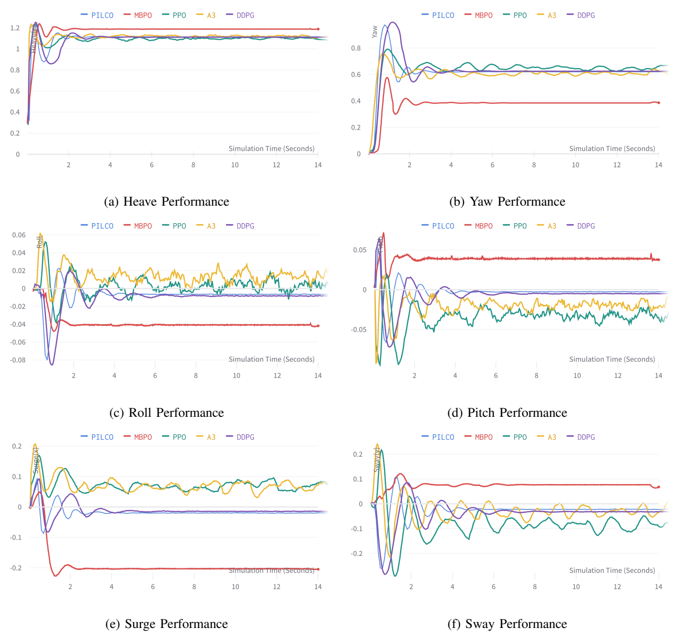
TABLE VII. TIME DOMAIN SPECIFICATIONS OF HEAVE AND YAW STATES OF FIVE RL ALGORITHMS

Algorithm	Specification	Heave	Yaw
PPO	Overshoot (%)	11.76	24.7
	Settling Time (s)	15.55	15.95
	Rise Time (s)	0.05	0.55
	Positioning Error	0.05 (cm)	0.16 (rad)
A3C	Overshoot (%)	9.61	25
	Settling Time (s)	15	15.45
	Rise Time (s)	0.01	0.35
	Positioning Error	0.03 (cm)	0.15 (rad)
DDPG	Overshoot (%)	4.25	50.8
	Settling Time (s)	1.09	13.9
	Rise Time (s)	0.41	0.67
	Positioning Error	0.02 (cm)	0.12 (rad)
MBPO	Overshoot (%)	4.26	52.08
	Settling Time (s)	3.18	7.23
	Rise Time (s)	1.11	1.86
	Positioning Error	0.1 (cm)	0.2 (rad)
PILCO	Overshoot (%)	12	56.25
	Settling Time (s)	0.65	1.05
	Rise Time (s)	0.01	0.15
	Positioning Error	0.01 (cm)	0.01 (rad)

Fig. 11 presents the action applied via each RL agent in every step of the running with a trained RL agent. MBPO always selects to go to the boundaries of action spaces. The boundary-seeking tendency exhibited by MBPO, stands as a crucial determinant in the suboptimal performance of MBPO across all scenarios. This behavior contributes to an elevated steady-state error in the final response. However, the other 4 algorithms are hovering around zero. The A3 and PPO have more erratic behavior than others. DDPG and PILCO have the smoothest learned action closer to zero.

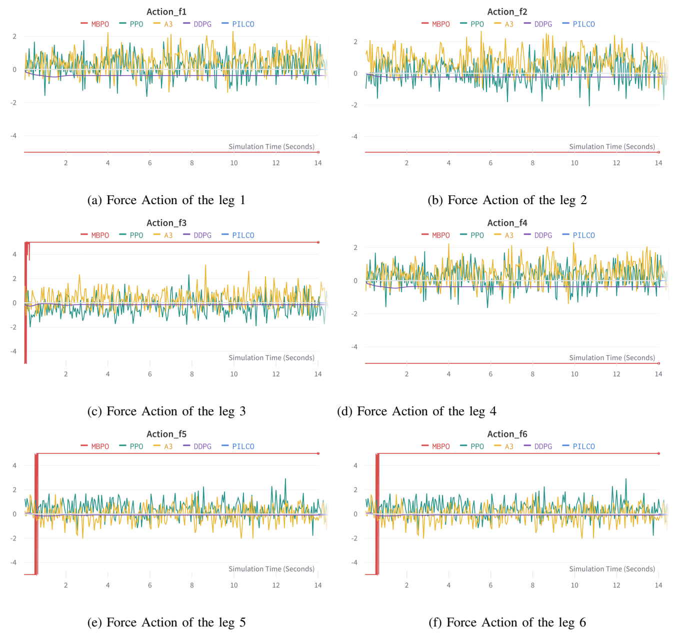
VI. CONCLUSION

In this study, we presented an RL topology for control of the Stewart platform in which we can learn and apply the



(a) Heave Performance (b) Yaw Performance (c) Roll Performance (d) Pitch Performance (e) Surge Performance (f) Sway Performance

Fig. 10. Time domain performances of the moving platform's states



(a) Force Action of the leg 1 (b) Force Action of the leg 2 (c) Force Action of the leg 3 (d) Force Action of the leg 4 (e) Force Action of the leg 5 (f) Force Action of the leg 6

Fig. 11. Legs force values of the five RL algorithms.

required force through RL agents in the reaching task as shown in Figure 7. We experimented with three model-free and two model-based RL algorithms. All five algorithms could learn to apply the required forces to boost the controller's performance. The key findings underscore the potential of RL algorithms in refining the control of the Stewart platform, with PILCO leading in terms of performance metrics. The application of RL, as evidenced in this study, holds promise for improving precision and adaptability in real-world scenarios. In conclusion, the PILCO algorithm has an overall better performance than other algorithms. The convergence of MBPO is notably the weakest, even though it shows more stability in the final steps compared to the other three model-free algorithms. In general, model-based algorithms tend to achieve more stable convergence than their model-free counterparts. However, all algorithms struggle in terms of both time performance and convergence when measured against PILCO. This Algorithm, grounded in probability and models, effectively handles system uncertainties, making it adaptable to dynamic environmental changes. This adaptability reduces rise time by allowing swift adjustments to control policies in evolving conditions. Additionally, uncertainty modeling minimizes steady-state errors, enhancing precision in maintaining the desired pose. Although PILCO is sample-efficient and has better and faster convergence performance in learning, it is the most computationally expensive algorithm.

Acknowledging the computational expense associated with PILCO, future research endeavors should prioritize optimizing computational efficiency without compromising performance. Also, expanding the experiment to encompass reaching an area rather than a precise pose could yield insightful findings about the platform's operational range. This broader exploration would provide a more comprehensive understanding of its capabilities and limitations. Moreover, conducting tests of the learned reaching task in similar scenarios would yield significant benefits. Engaging in such extensive testing would validate the applicability of the learned reaching task and set the stage for the platform's broader utilization in real-world applications. Furthermore, introducing trajectory following as a task can contribute to a deeper comprehension of how RL algorithms navigate and engage within the context of the Stewart platform environment. In future works, exploring this broader area of study could greatly improve our ability to control the Stewart platform. This might help us create better ways to control the platform for various applications. Finally, To validate learned behaviors in practice, extensive testing in controlled yet realistic settings is crucial. Simulating challenging conditions, such as turbulence in flight simulators (using the Stewart platform as a motion platform), can provide insights into the platform's adaptability. Collaborating with industry experts and conducting field trials in relevant environments will be instrumental in validating the learned behaviors and assessing the platform's effectiveness in addressing the complexities of real-world applications. Additionally, these findings may also have broader applicability to other robotic systems or real-world scenarios, such as serial manipulators, due to the similar duality that exists in those contexts.

References

[1] X. Yang, H. Wu, B. Chen, S. Kang, and S. Cheng, "Dynamic modeling and decoupled control of a flexible stewart platform for vibration isolation," *Journal of Sound and Vibration*, vol. 439, pp. 398–412, 2019, doi: 10.1016/J.JSV.2018.10.007.

[2] T. Ono, R. Eto, J. Yamakawa, and H. Murakami, "Analysis and control of a stewart platform as base motion compensators-part i: Kinematics using moving frames," *Nonlinear Dynamics*, vol. 107, pp. 51–76, 2022, doi: 10.1007/s11071-021-06767-8.

[3] A. Jishnu, D. K. Chauhan, and P. R. Vundavilli, "Design of neural network-based adaptive inverse dynamics controller for motion control of stewart platform," *International Journal of Computational Methods*, vol. 19, no. 08, p. 2142010, 2022, doi: 10.1142/s021987622142010x.

[4] S. O. Abioye, L. O. Oyedele, L. Akanbi, A. Ajayi, J. M. D. Delgado, M. Bilal, O. O. Akinade, and A. Ahmed, "Artificial intelligence in the construction industry: A review of present status, opportunities and future challenges," *Journal of Building Engineering*, vol. 44, p. 103299, 2021, doi: 10.1016/j.jobe.2021.103299.

[5] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," 2018, doi: 10.1109/TNN.1998.712192.

[6] Z. Ding, Y. Huang, H. Yuan, and H. Dong, "Introduction to reinforcement learning," *Deep reinforcement learning: fundamentals, research and applications*, pp. 47–123, 2020, doi: 10.1007/978-981-15-4095-0.

[7] S. P. Sethi and S. P. Sethi, "What is optimal control theory?," *Springer*, 2019.

[8] Y. Fang, Z. Huang, J. Pu, and J. Zhang, "Auv position tracking and trajectory control based on fast-deployed deep reinforcement learning method," *Ocean Engineering*, vol. 245, p. 110452, 2022.

[9] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013, doi: 10.1177/0278364913495721.

[10] M. I. Hosseini, S. A. Khalilpour, and H. D. Taghirad, "Practical robust nonlinear pd controller for cable-driven parallel manipulators," *Nonlinear Dynamics*, vol. 106, no. 1, pp. 405–424, 2021.

[11] M. Tipaldi, R. Iervolino, and P. R. Massenio, "Reinforcement learning in spacecraft control applications: Advances, prospects, and challenges," *Annual Reviews in Control*, 2022.

[12] I. A. Zamfirache, R.-E. Precup, R.-C. Roman, and E. M. Petriu, "Reinforcement learning-based control using q-learning and gravitational search algorithm with experimental validation on a nonlinear servo system," *Information Sciences*, vol. 583, pp. 99–120, 2022.

[13] L. Ghorbani and V. E. Omurlu, "Neural networks based real time solution for forward kinematics of a 6× 6 upu flight simulator," *Intelligent Service Robotics*, vol. 15, no. 5, pp. 611–626, 2022.

[14] J. A. Houck, R. J. Telban, and F. M. Cardullo, "Motion cueing algorithm development: Human-centered linear and nonlinear approaches," 2005.

[15] H. Sadjadian, H. D. Taghirad, and A. Fatehi, "Neural networks approaches for computing the forward kinematics of a redundant parallel manipulator," *International Journal of Computer and Information Engineering*, vol. 2, no. 1, pp. 1664–1671, 2008.

[16] Z. E. Kuzeci, V. E. Omurlu, H. Alp, and I. Ozkol, "Workspace analysis of parallel mechanisms through neural networks and genetic algorithms," in *2012 12th IEEE International Workshop on Advanced Motion Control (AMC)*, pp. 1–6, 2012, doi: 10.1109/AMC.2012.6197147.

[17] H. Yadavari, V. Tavakol Aghaei, and S. İkizoğlu, "Deep reinforcement learning-based control of stewart platform with parametric simulation in ros and gazebo," *Journal of Mechanisms and Robotics*, vol. 15, no. 3, p. 035001, 2023, doi: 10.1115/1.4056971.

[18] S. Pedrammehr, B. Danaei, H. Abdi, M. T. Masouleh, and S. Nahavandi, "Dynamic analysis of hexarot: axis-symmetric parallel manipulator," *Robotica*, vol. 36, no. 2, pp. 225–240, 2018, doi: 10.1017/S0263574717000315.

[19] S. Staicu, "Dynamics of Parallel Robots," *Springer*, 2019.

[20] A. Arian, B. Danaei, and M. Tale Masouleh, "Kinematic and dynamic analyses of tripteron, an over-constrained 3-dof translational parallel manipulator, through newton-euler approach," *AUT Journal of Modeling and Simulation*, vol. 50, no. 1, pp. 61–70, 2018, doi: 10.22060/MISCJ.2018.13020.5055.

[21] S. Pakzad, S. Akhbari, and M. Mahboubkhah, "Kinematic and dynamic analyses of a novel 4-dof parallel mechanism," *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, vol. 41, pp. 1–13, 2019, doi: 10.1007/s40430-019-2058-3.

[22] R. F. Abo-Shanab, "Dynamic modeling of parallel manipulators based on lagrange–d'alembert formulation and jacobian/hessian matrices," *Multibody System Dynamics*, vol. 48, no. 4, pp. 403–426, 2020, doi: 10.1007/s11044-019-09705-0C.

[23] S. Chen, G. Cheng, and Y. Pang, "Dynamic analysis and trajectory tracking control for a parallel manipulator with joint friction," *Applied Sciences*, vol. 12, no. 13, p. 6682, 2022, doi: 10.3390/app12136682.

[24] L.-W. Tsai, "Solving the inverse dynamics of a stewart-gough manipulator by the principle of virtual work," *J. Mech. Des.*, vol. 122, no. 1, pp. 3–9, 2000, doi: 10.1115/1.533540.

[25] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, "Sympy: symbolic computing in python," *PeerJ Computer Science*, vol. 3, p. e103, 2017, doi: 10.7717/peerj-cs.103.

[26] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020, doi: 10.1038/s41586-020-2649-2.

[27] I. Faktorovich, C. Bohn, and J. Vogelsang, "Reinforcement learning motivated feedforward control approach for disturbance rejection and tracking," in *2021 European Control Conference (ECC)*, pp. 138–143, 2021, doi: 10.23919/ecc54610.2021.9655160.

[28] J. Seo, Y.-S. Na, B. Kim, C. Lee, M. Park, S. Park, and Y. Lee, "Feedforward beta control in the kstar tokamak by deep reinforcement learning," *Nuclear Fusion*, vol. 61, no. 10, p. 106010, 2021, doi: 10.1088/1741-4326/ac121b.

[29] R. Liu, F. Nageotte, P. Zanne, M. de Mathelin, and B. Dresp-Langley, "Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review," *Robotics*, vol. 10, no. 1, p. 22, 2021, doi: 10.3390/robotics10010022.

[30] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[31] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, 2015.

[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[33] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.

[34] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," *Advances in neural information processing systems*, vol. 32, 2019.

[35] X. Li, L. Dong, L. Xue, and C. Sun, "Hybrid reinforcement learning for optimal control of non-linear switching system," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 11, pp. 9161–9170, 2023, doi: 10.1109/TNNLS.2022.3156287.

[36] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, "Benchmarking model-based reinforcement learning," *Computer Science*, 2019.

[37] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, 2018.

[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015, doi: 10.1038/nature14236.

[39] F. Xue, Q. Hai, T. Dong, Z. Cui, and Y. Gong, "A deep reinforcement learning based hybrid algorithm for efficient resource scheduling in edge computing environment," *Information Sciences*, vol. 608, pp. 362–374, 2022, doi: 10.1016/j.ins.2022.06.078.

[40] S. Kizir and Z. Bingül, "Design and development of a stewart platform assisted and navigated transsphenoidal surgery," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 27, no. 2, pp. 961–972, 2019, doi: 10.3906/ELK-1608-145.

[41] Z. Bingul and O. Karahan, "Dynamic modeling and simulation of stewart platform," 2012, doi:10.5772/32470.

[42] D. Rawat, M. K. Gupta, and A. Sharma, "Intelligent control of robotic manipulators: a comprehensive review," *Spatial Information Research*, vol. 31, no. 3, pp. 345–357, 2023, doi: 10.1007/s41324-022-00500-2.

[43] V. S. D. M. Sahu, P. Samal, and C. K. Panigrahi, "Modelling, and control techniques of robotic manipulators: A review," *Materials Today: Proceedings*, vol. 56, pp. 2758–2766, 2022, doi: 10.1016/j.matpr.2021.10.009.

[44] A. P. Singh, D. Deb, H. Agrawal, K. Bingi, and S. Ozana, "Modeling and control of robotic manipulators: A fractional calculus point of view," *Arabian Journal for Science and Engineering*, vol. 46, no. 10, pp. 9541–9552, 2021, doi: 10.1007/s13369-020-05138-6.

[45] Z. Liu, K. Peng, L. Han, and S. Guan, "Modeling and control of robotic manipulators based on artificial neural networks: a review," *Iranian Journal of Science and Technology, Transactions of Mechanical Engineering*, pp. 1–41, 2023, doi: 10.1007/s40997-023-00596-3.

[46] M. Engin, "Controller design for parallel mechanism solar tracker," *Machines*, vol. 11, no. 3, p. 372, 2023, doi: 10.3390/machines11030372.

[47] H. D. Taghirad, *Parallel robots: mechanics and control*. CRC press, 2013.

[48] H. Yadavari, "library to calculate the inverse dynamic problem of the stewart platform." Available: https://github.com/HadiYd/stewart_inverse_dynamic

[49] Gazebo Development Team, "Gazebo: An Open-Source Simulator for Robotics Research," http://gazebosim.org, 2023.

[50] Amazon Web Services. (2023) Amazon Elastic Compute Cloud (Amazon EC2). Amazon Web Services. [Online]. Available: https://aws.amazon.com/ec2/

[51] T.-T. Do, V.-H. Vu, and Z. Liu, "Linearization of dynamic equations for vibration and modal analysis of flexible joint manipulators," *Mechanism and Machine Theory*, vol. 167, p. 104516, 2022, doi: 10.1016/J.MECHMACHTHEORY.2021.104516.

[52] A. A. Kumar, J.-F. Antoine, and G. Abba, "Input-output feedback linearization for the control of a 4 cable-driven parallel robot," *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 707–712, 2019, doi: 10.1016/j.ifacol.2019.11.154.

[53] K. Senda, S. Mano, and S. Fujii, "A reinforcement learning accelerated by state space reduction," in *SICE 2003 Annual Conference (IEEE Cat. No. 03TH8734)*, vol. 2, pp. 1992–1997, 2003.

[54] T. Sadamoto, A. Chakrabortty, and J.-i. Imura, "Fast online reinforcement learning control using state-space dimensionality reduction," *IEEE Transactions on Control of Network Systems*, vol. 8, no. 1, pp. 342–353, 2020, doi: 10.1109/TCNS.2020.3027780.

[55] V. T. Aghaei, A. Ağababaoğlu, S. Yıldırım, and A. Onat, "A real-world application of markov chain monte carlo method for bayesian trajectory control of a robotic manipulator," *ISA transactions*, vol. 125, pp. 580–590, 2022, doi: 10.1016/j.isatra.2021.06.010.

[56] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[57] N. D. Nguyen, T. T. Nguyen, P. Vamplew, R. Dazeley, and S. Nahavandi, "A prioritized objective actor-critic method for deep reinforcement learning," *Neural Computing and Applications*, vol. 33, pp. 10335–10349, 2021, doi: 10.1007/s00521-021-05795-0.

[58] Y. Yuan, K. Dehghanpour, Z. Wang, and F. Bu, "A joint distribution system state estimation framework via deep actor-critic learning method," *IEEE Transactions on Power Systems*, vol. 38, no. 1, pp. 796–806, 2022, doi: 10.1109/TPWRS.2022.3155649.

[59] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*, pp. 387–395, 2014.

[60] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, 2015.

[61] O. O. Abayomi-Alli, R. Damaševičius, A. Qazi, M. Adedoyin-Olowe, and S. Misra, "Data augmentation and deep learning methods in sound classification: A systematic review," *Electronics*, vol. 11, no. 22, p. 3795, 2022, doi: 10.3390/electronics11223795.

[62] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017, doi: 10.1145/3065386.

[63] A. Mumuni and F. Mumuni, "Data augmentation: A comprehensive survey of modern approaches," *Array*, p. 100258, 2022, doi: 10.1016/j.array.2022.100258.

[64] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Machine learning proceedings 1990*, pp. 216–224, 1990, doi: 10.1016/b978-1-55860-141-3.50030-4.

[65] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 2, pp. 408–423, 2013, doi: 10.1109/TPAMI.2013.218.

Appendix A
Deriving Dynamic Formulation

For consistency with the numbering of the equations, the generalized coordinates of the system are defined as $(l_1, l_2, l_3, l_4, l_5, l_6)$, which correspond to the six joints of the platform. Each link in the system is associated with a reference frame that is attached to it. The two most essential reference frames are the base (B) and moving platform (M) frames. Their orientation matrix is defined as Equation (15). Here, the angles β, α, and γ represent successive rotations respectively near the x-axis, y-axis, and z-axis.

$$\begin{bmatrix} c(\beta(t))c(\gamma(t)) & s(\alpha(t))s(\beta(t))c(\gamma(t)) - s(\gamma(t))c(\alpha(t)) & s(\alpha(t))s(\gamma(t)) + s(\beta(t))c(\alpha(t))c(\gamma(t)) \\ s(\gamma(t))c(\beta(t)) & s(\alpha(t))s(\beta(t))s(\gamma(t)) + c(\alpha(t))c(\gamma(t)) & -s(\alpha(t))c(\gamma(t)) + s(\beta(t))s(\gamma(t))c(\alpha(t)) \\ -s(\beta(t)) & s(\alpha(t))c(\beta(t)) & c(\alpha(t))c(\beta(t)) \end{bmatrix} \tag{15}$$

The platform pose, speed, and acceleration are defined in Equations (16),(17), (18):

$$X = [P_x(t),\ P_y(t),\ P_z(t),\ \alpha(t),\ \beta(t),\ \gamma(t)] \tag{16}$$

$$\dot{X} = \left[\dot{P}_x,\ \dot{P}_y,\ \dot{P}_z,\ \dot{\alpha},\ \dot{\beta},\ \dot{\gamma}\right] \tag{17}$$

$$\ddot{X} = \left[\ddot{P}_x,\ \ddot{P}_y,\ \ddot{P}_z,\ \ddot{\alpha},\ \ddot{\beta},\ \ddot{\gamma}\right] \tag{18}$$

Equation (19) from [24] is utilized to compute the velocity of the center of the ball joint B_i:

$$v_{bi} = v_p + \omega_p \times b_i \tag{19}$$

Equations (20), (21) are employed to calculate the linear and angular velocity of the piston relative to the cylinder:

$$\dot{d}_i =^i v_{bi,z} \tag{20}$$

$$^i\omega_i = \frac{1}{d_i}(^is_i \times\ ^iv_{bi}) = \frac{1}{d_i}\begin{bmatrix} -^iv_{bi,y} \\ ^iv_{bi,x} \\ 0 \end{bmatrix} \tag{21}$$

The velocity of the cylinder $(^iv_{1i})$ and pistons' centers of mass $(^iv_{2i})$ are calculated as in (22),(23):

$$^iv_{1i} = e_1^i\omega_i \times\ ^is_i = \frac{e_1}{d_i}\begin{bmatrix} ^iv_{bi,x} \\ ^iv_{bi,y} \\ 0 \end{bmatrix} \tag{22}$$

$$^iv_{2i} = (d_i - e_2)^i\omega_i \times^i s_i + \dot{d}_i\,^is_i = \frac{e_1}{d_i}\begin{bmatrix} (d_i - e_2)^iv_{bi,x} \\ (d_i - e_2)^iv_{bi,y} \\ d_i^iv_{bi,y} \end{bmatrix} \tag{23}$$

The Jacobian matrix does the necessary transformation required in finding the actuator forces from the moving platform's forces and moments, besides the velocities [47].

The matrix form of Equation (19), denoted as Equation (24), is expressed as follows:

$$v_{bi} = J_{bi}\dot{X}_p \tag{24}$$

In Equation (24), the vector $\dot{X}p$ represents the linear and angular velocities of the moving platform, and Jbi is defined according to Equation (25).

$$J_{bi} = \begin{bmatrix} 1 & 0 & 0 & 0 & b_{i,z} & -b_{i,y} \\ 0 & 1 & 0 & -b_{i,z} & 0 & b_{i,x} \\ 0 & 0 & 1 & b_{i,y} & -b_{i,x} & 0 \end{bmatrix} \tag{25}$$

We can write Equation (20) like Eqation (26) as below:

$$\dot{d}_i =^i J_{bi,z}\dot{X}_p \tag{26}$$

Repeating Equation (26) six times for each leg, we have the equations in matrix form represented in Equation (27).

$$\dot{q} = J_p\dot{X}_p \tag{27}$$

where

$$J_p = \begin{bmatrix} ^1J_{b1,z} \\ ^2J_{b2,z} \\ ^3J_{b3,z} \\ ^4J_{b4,z} \\ ^5J_{b5,z} \\ ^6J_{b6,z} \end{bmatrix} \tag{28}$$

Equation (28) is known as the manipulator Jacobian matrix. Also, we can write Equations (21), (22) and (23) as:

$$^{i}\omega_i = \frac{1}{d_i}\begin{bmatrix} -^{i}J_{bi,y} \\ ^{i}J_{bi,x} \\ 0_{1x6} \end{bmatrix}\dot{X}_p \tag{29}$$

and,

$$^{i}v_{1i} = \frac{e_1}{d_i}\begin{bmatrix} ^{i}J_{bi,x} \\ ^{i}J_{bi,y} \\ 0_{1x6} \end{bmatrix}\dot{X}_p \tag{30}$$

and,

$$^{i}v_{2i} = \frac{1}{d_i}\begin{bmatrix} (d_i - e_2)^{i}J_{bi,x} \\ (d_i - e_2)^{i}J_{bi,y} \\ d_i^{i}J_{bi,y} \end{bmatrix}\dot{X}_p \tag{31}$$

Then, if we combine the Equations (29) , (30) and (31) we have :

$$^{i}\dot{X}_{1i} =^{i} J_{1i}\dot{X}_p \tag{32}$$

$$^{i}\dot{X}_{2i} =^{i} J_{2i}\dot{X}_p \tag{33}$$

where the link Jacobian matrices are Equations (34) , (35):

$$^{i}J_{1i} = \frac{1}{d_i}\begin{bmatrix} e_1^{i}J_{bi,x} \\ e_1^{i}J_{bi,y} \\ 0_{1x6} \\ -^{i}J_{bi,y} \\ ^{i}J_{bi,x} \\ 0_{1x6} \end{bmatrix} \tag{34}$$

$$^{i}J_{2i} = \frac{1}{d_i}\begin{bmatrix} (d_i - e_2)^{i}J_{bi,x} \\ (d_i - e_2)^{i}J_{bi,y} \\ d_i^{i}J_{bi,z} \\ -^{i}J_{bi,y} \\ ^{i}J_{bi,x} \\ 0_{1x6} \end{bmatrix} \tag{35}$$

With the availability of these Jacobian matrices, we can now proceed to solve the inverse dynamics problem, which involves determining the forces required to achieve a desired motion. As outlined in [24], the equations of motion are formulated using the principle of virtual work.

Equation (36) demonstrates the external and inertia forces acting on the center of mass of the moving platform.

$$F_P = \begin{bmatrix} \hat{f}_p \\ \hat{n}_p \end{bmatrix} = \begin{bmatrix} f_e + m_p g - m_p \dot{v}_p \\ n_e -^{A} I_p\dot{\omega}_p - \omega_p \times (^{A}I_p\omega_p) \end{bmatrix} \tag{36}$$

In Equation (36), f_e and n_e represent the external force and moment, respectively, applied at the center of mass of the moving platform [24]. Similarly, we can consider the same forces for the cylinder and piston of each leg, assuming that the gravitational force is the only external force present:

$$^{i}F_{1i} = \begin{bmatrix} \hat{f}_{1i} \\ \hat{n}_{1i} \end{bmatrix} = \begin{bmatrix} m_{1i}\,^{i}R_A g - m_{1i}\,^{i}\dot{v}_{1i} \\ -^{i}I_{1i}\,^{i}\dot{\omega}_i -^{i} \omega_i \times (^{i}I_{1i}^{i}\omega_i) \end{bmatrix} \tag{37}$$

$$^{i}F_{2i} = \begin{bmatrix} \hat{f}_{2i} \\ \hat{n}_{2i} \end{bmatrix} = \begin{bmatrix} m_{2i}\,^{i}R_A g - m_{2i}\,^{i}\dot{v}_{1i} \\ -^{i}I_{2i}\,^{i}\dot{\omega}_i -^{i} \omega_i \times (^{i}I_{2i}^{i}\omega_i) \end{bmatrix} \tag{38}$$

The principle of virtual work is stated as below:

$$\delta q^T\tau + \delta X_p^T F_p + \sum_i^6(\delta\,^{i}X_{1i}^T\,^{i}F_{1i} + \delta\,^{i}X_{2i}^T\,^{i}F_{2i}) = 0 \tag{39}$$

where in the leg frame, we have the applied and inertia forces, $^{i}F_{1i}$ and $^{i}F_{2i}$, and their corresponding virtual displacements $\delta\,(^{i}X_{1i})$ and $\delta\,(^{i}X_{2i})$.

To establish a relationship between these virtual displacements, we need to express them in terms of a set of generalized virtual displacements, which can be defined as follows:

$$\delta q = J_p\delta X_p \tag{40}$$

$$\delta^{i}X_{1i} =^{i} J_{1i}\,\delta X_p \tag{41}$$

$$\delta^{i}X_{2i} =^{i} J_{2i}\,\delta X_p \tag{42}$$

By substituting Equations (40), (41), and (42) into Equation (39), we can derive the dynamics of the Stewart platform as follows:

$$J_p^T\tau + F_p + \sum_i^6(\delta\,^{i}X_{1i}^T\,^{i}F_{1i} + \delta\,^{i}X_{2i}^T\,^{i}F_{2i}) = 0 \tag{43}$$

Then,

$$J_p^T(\tau + F_z) + F_p + J_x^T F_x + J_y^T F_y = 0 \tag{44}$$

If J_p is not singular, we can have the final inverse dynamic as shown in Equation (45):

$$\bar{\tau} = -F_z - J_p^{-T}(F_p + J_x^T F_x + J_y^T F_y) \tag{45}$$