WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Trajectory Generation in Robotics

Ms. Priyanka Chemudugunta[1], Madhan. E [2], S. Ranjith[3], K. M. Reyashini[4]

[1]Assistant Professor, Department Of Robotics And Automation Engineering, Karpaga Vinayaga College Of Engineering And Technology, Chengalpattu District,Tamilnadu,India.

[2,3,4]Undergraduate student, Department Of Robotics And Automation Engineering, Karpaga Vinayaga College Of Engineering And Technology, Chengalpattu District, Tamilnadu, India.

A B S T R A C T

Trajectory generation in robotics is a fundamental process that determines the precise path a robot must follow to move smoothly from one point to another. This process involves computing both the spatial and temporal evolution of the robot's motion, ensuring that all kinematic and dynamic constraints are met. Trajectories are typically generated either in joint space, where the motion of individual robot joints is controlled, or in Cartesian space, where the overall position of the robot's end-effector is regulated. The goal of trajectory generation is to ensure smooth, accurate, and efficient motion, which is critical for a wide range of robotic applications, including industrial automation, autonomous vehicles, drones, and service robots. Various mathematical models are used to generate these trajectories, such as cubic polynomials, higher-order splines, and linear segments with parabolic blends. Each of these models ensures that the position, velocity, and acceleration of the robot's joints or end-effector are continuously controlled over time. Time-optimal trajectory planning, minimum-jerk, and minimum torque approaches are also employed to improve performance in terms of speed and energy efficiency. However, trajectory generation is not without its challenges.

Keywords: Trajectory generation, Polynomial trajectories, Kinematic constraints, Dynamic constraints, A* algorithm

Introduction

Trajectory generation is one of the key elements in robotic motion planning and control, ensuring that robots can move from one point to another in a smooth, precise, and efficient manner. It involves the process of computing the desired positions, velocities, and accelerations over time for each part of a robot to follow a predetermined path. This computation must consider various factors, including the robot's kinematic and dynamic constraints, task specific requirements, and environmental limitations. Robotic systems, whether they are robotic arms, mobile robots, or autonomous vehicles, need to follow specific trajectories to perform tasks such as pick-and-place operations, welding, painting, or navigation. The complexity of these tasks varies based on the type of robot and its environment. For example, industrial robots performing repetitive tasks in controlled environments might require relatively simple trajectory planning. In contrast, mobile robots operating in dynamic and unpredictable environments must generate complex, real time trajectories that allow them to navigate safely and efficiently. There are two primary types of trajectory generation used in robotics: joint-space trajectories and Cartesian-space trajectories[1]. In joint-space trajectory generation, the motion is described in terms of the individual joint angles of the robot, with the goal of ensuring smooth transitions between target positions while respecting joint constraints.

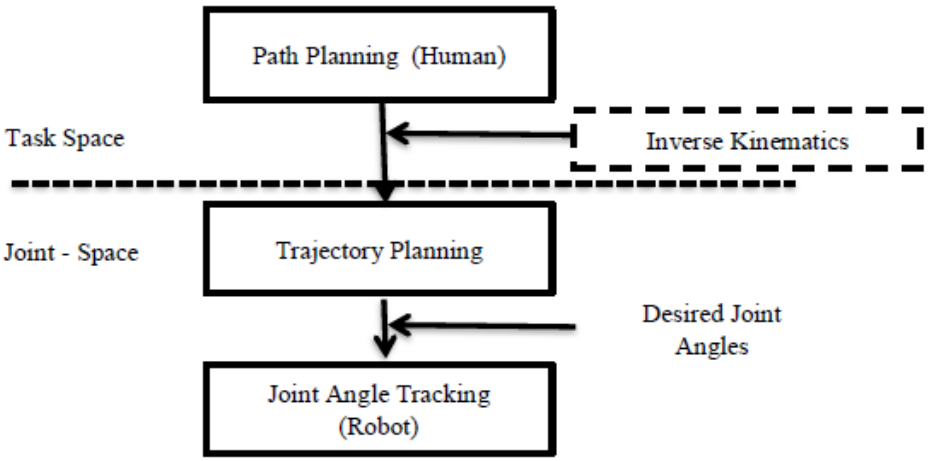

Figure 1.Representation Of Trajectory Planning

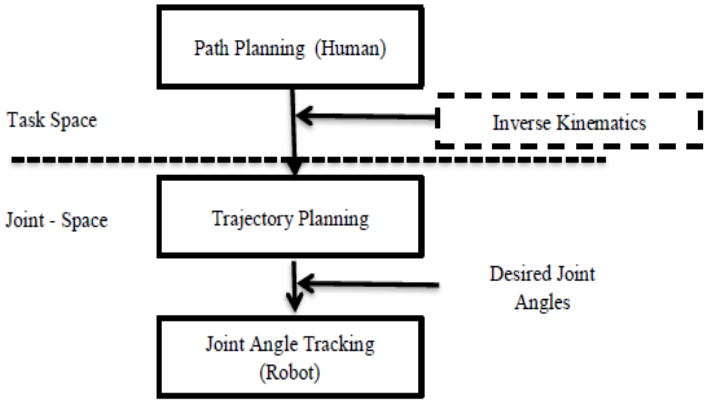

Figure 1.Representation Of Trajectory Planning

In Cartesian-space trajectory generation, the focus is on controlling the position and orientation of the robot's end-effector in the global coordinate system. This method is particularly useful when the robot's task involves interacting with objects or environments in specific spatial locations. The mathematical foundations of trajectory generation rely on techniques such as cubic polynomials, linear segments with parabolic blends, and higher-order splines. These methods are chosen for their ability to produce smooth transitions in motion, which are crucial to avoiding abrupt movements that could cause mechanical wear or reduce the accuracy of the task being performed. In practice, trajectory generation must address several constraints. Kinematic constraints include limits on joint angles, velocities, and accelerations, while dynamic constraints are related to the physical capabilities of the robot's actuators, such as the maximum torque they can produce. Additionally, environmental constraints like obstacles, boundaries, and other dynamic elements must be considered, particularly for mobile robots and autonomous systems.[2]

To handle these challenges, various algorithms have been developed, including the A algorithm*, D algorithm*, and Rapidly-exploring Random Trees (RRT). These algorithms enable robots to plan and generate feasible trajectories in both static and dynamic environments. Advances in real-time processing capabilities and sensor technologies have further enhanced the ability of robots to generate adaptive trajectories that respond to changes in their surroundings. The importance of trajectory generation extends beyond industrial applications. Autonomous vehicles, drones, and service robots also rely heavily on robust trajectory generation techniques. For example, autonomous vehicles must generate trajectories that allow them to navigate safely through traffic, avoid obstacles, and obey traffic laws.

Similarly, drones use trajectory generation to maintain stable flight and execute complex maneuvers. In this journal, we will explore the detailed process of trajectory generation in robotics, the different approaches used for both joint-space and Cartesian-space trajectories, and the challenges associated with real-time trajectory planning. We will also discuss the mathematical models, algorithms, and techniques that enable efficient and reliable trajectory generation in various robotic systems. Finally, we will look at the future trends and innovations in trajectory generation, including the role of artificial intelligence and machine learning in enhancing robotic autonomy. Efficient trajectory generation is essential for ensuring the robot can operate safely

and effectively in its environment. [3]It impacts the performance of tasks such as pick-and-place operations, welding, assembly, and autonomous navigation. The accuracy and smoothness of trajectories are crucial for preventing mechanical wear and achieving optimal task execution.

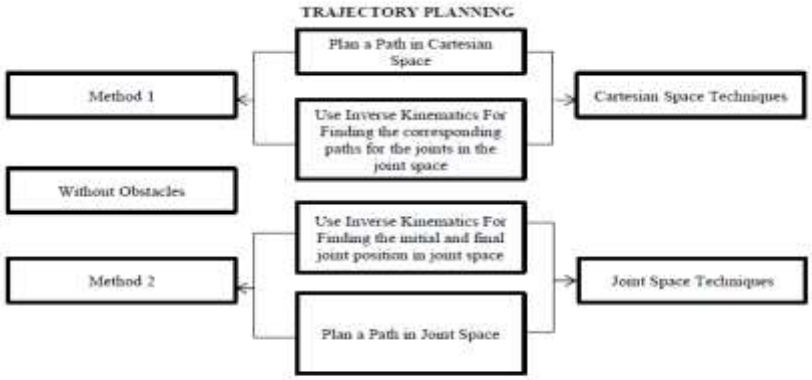

Figure 2. Methods In Trajectory Planning

METHOD

Types Of Trajectories In Robotics

In robotics, trajectory generation plays a crucial role in defining the motion of a robot from an initial to a target position. The path a robot follows, including its velocity and acceleration at any given point, must be carefully planned to ensure precise and smooth motion. Depending on the specific application and task requirements, two main types of trajectories are commonly used in robotics joint-space trajectories and Cartesian-space trajectories**.** Each type is suited to different tasks and offers distinct advantages in terms of control and execution.

Joint – Space Trajectory Planning

A joint-space trajectory defines the motion of a robot by specifying the desired positions, velocities, and accelerations of each of the robot's individual joints over time. In this type of trajectory, the focus is on controlling the movement of each joint directly, with the goal of ensuring smooth transitions from one joint configuration to another.

Key Characteristics Of Joint – Space Trajectory Planning

Joint Angle Control

In joint-space trajectories, the position of each joint is controlled independently. This method is ideal for robotic systems where precise joint positioning is critical, such as robotic arms in industrial settings.

Ease of Implementation

Joint-space trajectories are easier to compute and implement because they do not require complex calculations involving the end-effector's position in the global workspace. Each joint can be controlled directly, making this approach computationally efficient.[4].

Suitability for Robotic Arms

This method is commonly used in robotic manipulators, where the goal is to control each joint to move the end-effector to the desired location.[5] For example, in a pick-and-place operation, the robot moves from one joint configuration to another, ensuring that the end-effector reaches the target object accurately.

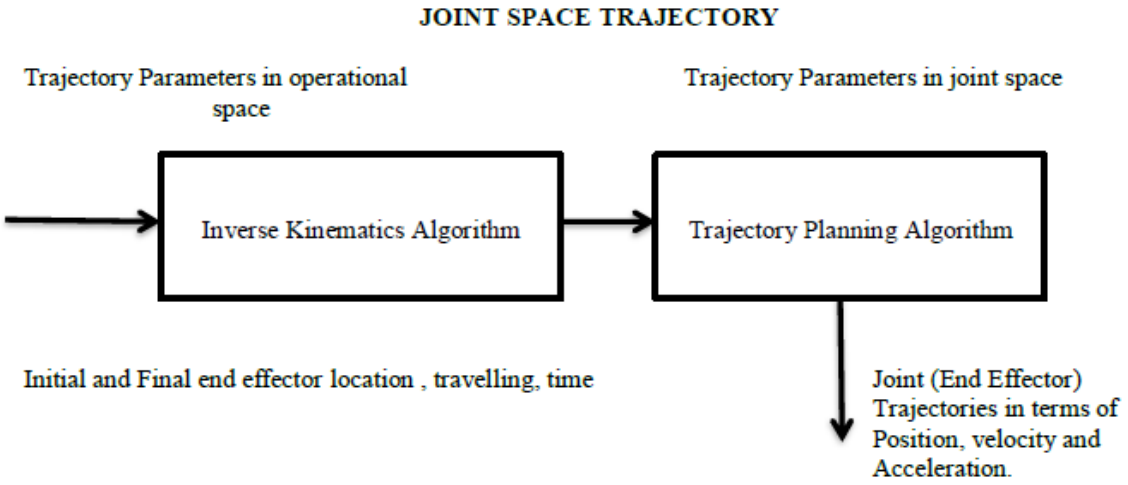

Figure 3. Representation Of Joint Space Trajectory Planning

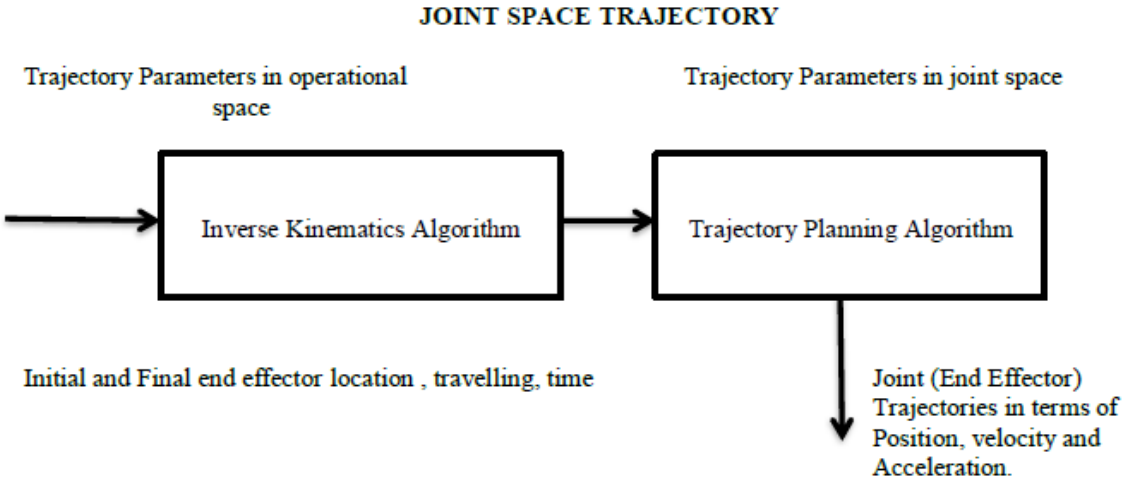

Figure 3. Representation Of Joint Space Trajectory Planning

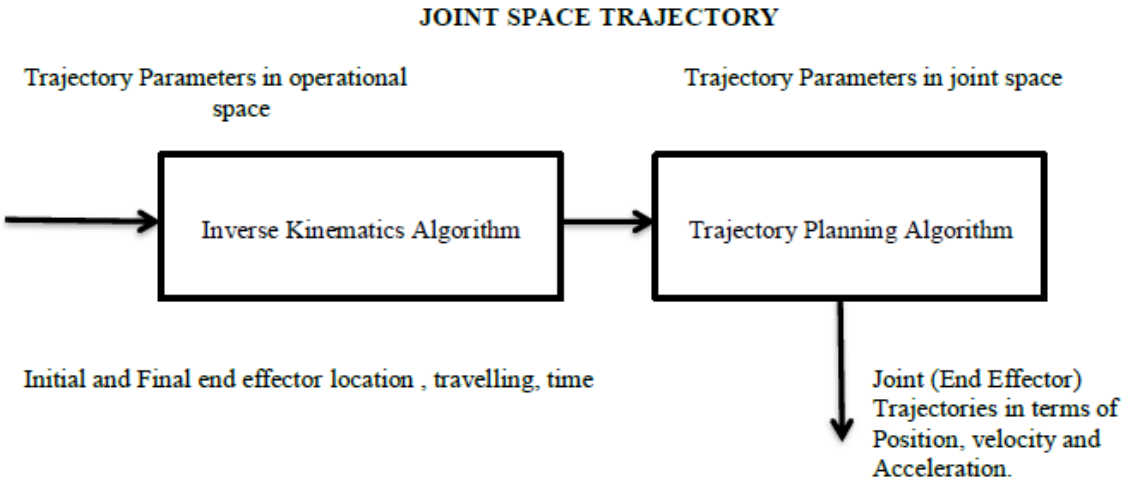

Figure 3. Representation Of Joint Space Trajectory Planning

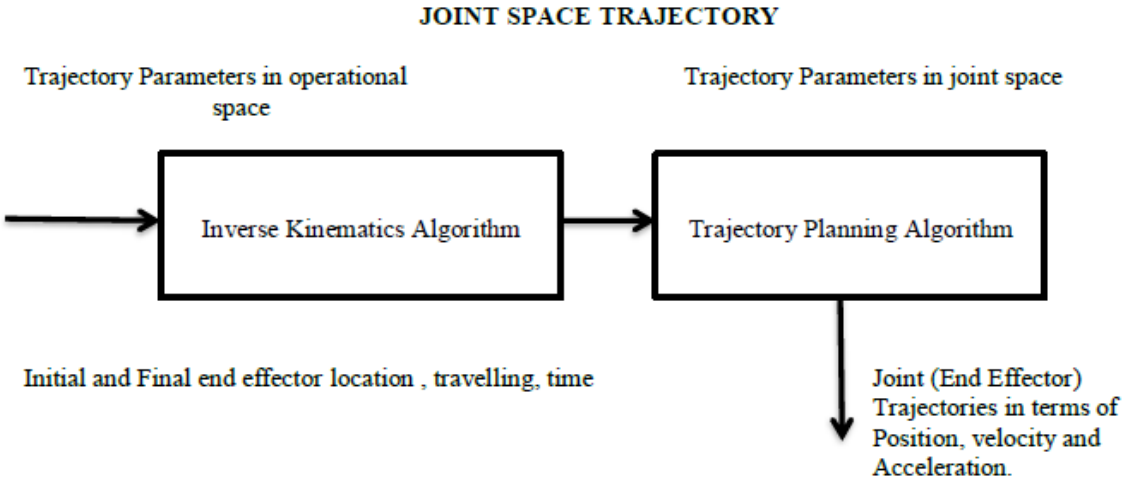

Figure 3. Representation Of Joint Space Trajectory Planning

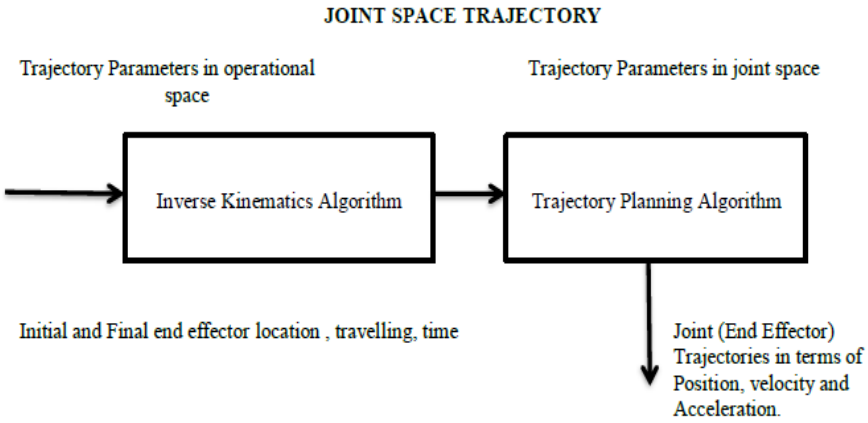

Figure 3. Representation Of Joint Space Trajectory Planning

One of the defining features of soft robotics is its inspiration drawn from natural systems, particularly biological organisms. By mimicking the flexibility and versatility found in living organisms, soft robots are capable of performing tasks that are challenging for conventional robots. For example, soft robotic grippers can delicately grasp irregularly shaped objects without causing damage, making them ideal for applications in food handling, agriculture, and manufacturing. By combining the principles of flexibility, adaptability, and biomimicry, soft robots are poised to revolutionize the way we interact with technology and navigate the world around us .

Benefits

Simplified Control

Since the trajectory is defined in terms of joint angles, the complexity of solving inverse kinematics (finding joint angles that produce a desired end-effector position) is often reduced or eliminated.

Smooth Transitions

Joint-space trajectories ensure smooth transitions in each joint, avoiding jerky or sudden movements that could damage the robot or reduce accuracy.

Limitations

End-Effector Path Uncertainty

While joint-space trajectories offer smooth joint transitions, the resulting path of the end-effector in Cartesian space may not be a straight line or the most efficient path. [6] This can be problematic in applications where precise positioning of the end-effector in the workspace is required.

Complex Mapping to Cartesian Space

In some cases, converting joint-space trajectories to Cartesian-space movements can lead to complicated, nonlinear behavior, making it difficult to predict the exact motion of the robot's end-effector.

Cartesian Space Trajectory Planning

A Cartesian-space trajectory specifies the motion of the robot's end-effector directly in terms of its position and orientation in a global coordinate system (x, y, z). This type of trajectory is particularly useful for tasks where the robot must interact with objects in the environment, such as assembly operations or object manipulation.[7]

Key Characteristics

End-Effector Control

In Cartesian-space trajectories, the position, velocity, and acceleration of the robot's end-effector are controlled directly in the workspace. This allows for precise control of the robot's interactions with objects in its environment.

Direct Path Planning

Cartesian trajectories are useful when the robot must follow a specific path in space, such as a straight line, circular arc, or complex 3D curve. For example, in welding or painting applications, the robot's end-effector must follow the exact contour of the surface being worked on.

Inverse Kinematics

To implement Cartesian-space trajectories, inverse kinematics calculations are required. These calculations determine the joint configurations necessary to achieve the desired end-effector position and orientation. This process can be computationally expensive, especially for robots with many degrees of freedom.[8]

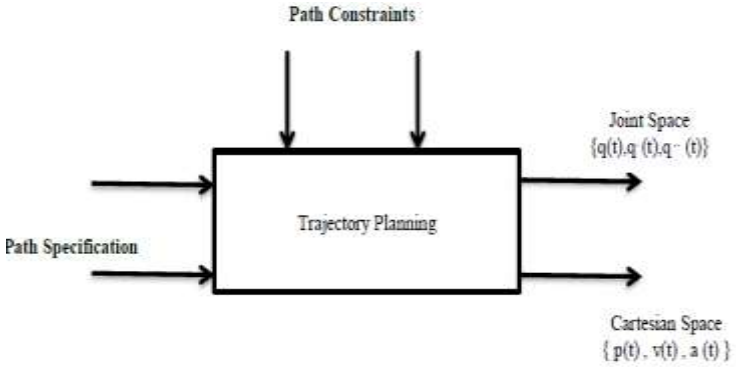

Figure 4. Representation Of Cartesian Space Trajectory Planning

Benefits

Precise Path Control

Cartesian-space trajectories provide greater control over the robot's path in the workspace, ensuring that the end-effector follows the desired trajectory with high precision. This is critical in tasks such as surface inspection, where the end-effector must move smoothly over an object without deviating from its path.

Efficiency in Task Execution

By defining trajectories in Cartesian space, robots can optimize their movements to complete tasks more efficiently. For example, a robot performing a painting operation can minimize unnecessary joint movements while following a smooth, continuous path across the surface.

Limitations

Complexity in Inverse Kinematics

One of the main challenges of Cartesian-space trajectory generation is solving the inverse kinematics problem. [9]This can be computationally complex and may result in multiple possible joint configurations for a given end-effector position.

Joint Limitations

The calculated joint movements needed to achieve the desired Cartesian path may sometimes exceed the physical limitations of the robot's joints, leading to potential issues with joint velocities, torques, or workspace boundaries.

Computational Overhead

Due to the need for real-time inverse kinematics and dynamic calculations, Cartesian-space trajectories can impose a significant computational burden, especially for robots with a large number of degrees of freedom.

Trajectory Generation

Trajectory generation in robotics involves creating smooth, continuous paths that a robot can follow to achieve its motion objectives while satisfying constraints such as velocity, acceleration, and dynamic limits. To achieve this, a range of mathematical models and techniques are applied. These models ensure that the robot can move safely and efficiently while adhering to its physical limitations. The mathematical foundation of trajectory generation is rooted in concepts like polynomials, splines, and optimization techniques, which are employed to generate position, velocity, and acceleration profiles.[10]

Polynomial Trajectories

Polynomial functions are widely used in trajectory generation because of their simplicity and ability to provide smooth transitions. They allow the designer to control the robot's position, velocity, and acceleration at different time points by solving for coefficients in polynomial equations. Polynomial trajectories typically use cubic or quintic polynomials**,** depending on the task requirements.

Cubic Polynomial Trajectory

Cubic polynomials are commonly used to generate simple trajectories that ensure continuous position and velocity. The general form of a cubic polynomial is

$$q(t)=a0+a1t+a2t^2+a3t^3 \quad (1)$$

Here, q(t) represents the joint position at time t and a0,a1,a2,a3 are the polynomial coefficients to be determined. To define a trajectory between two points, initial and final positions (qo and qf) and velocities ($q\dot{}0$ and $q\dot{}f$) are required. These boundary conditions allow the calculation of the coefficients by solving a system of equations. For a cubic polynomial, the position and velocity continuity are enforced at the start and end of the motion. Advantages of cubic polynomials include ease of computation and smooth transitions in position and velocity. However, they may not guarantee continuous acceleration, which is critical for more sensitive applications.[11]

Quintic Polynomial Trajectory

Quintic polynomials add higher-order terms to ensure continuous acceleration in addition to position and velocity

$$q(t)=a0+a1t+a2t^2+a3t^3+a4t^4+a5t^5 \quad (2)$$

For quintic polynomials, boundary conditions also include initial and final accelerations leading to a smoother overall trajectory. Quintic polynomials are especially useful in applications requiring more refined motion control, such as in surgical robots or highly dynamic systems. The choice between cubic and quintic polynomials depends on the smoothness requirements of the task and the computational resources available. [12] While cubic polynomials are easier to compute, quintic polynomials offer smoother transitions and are more suitable for tasks where acceleration continuity is crucial.

Linear Segments With Parabolic Blends

Linear Segments with Parabolic Blends (LSPB) is a widely used trajectory generation method in robotics that ensures smooth transitions between different motion phases. It combines linear motion segments with parabolic acceleration and deceleration phases, creating a trajectory that smoothly accelerates to a constant velocity and then decelerates to a stop[13]. The parabolic blends at the beginning and end of the motion ensure that the velocity starts and ends at zero, preventing abrupt changes and reducing mechanical stress on the robot. The linear segment allows the robot to move at a constant velocity, optimizing efficiency in tasks requiring sustained motion over long distances. This method is particularly effective in industrial automation applications, such as pick-and-place operations, where precise and smooth transitions between motion states are essential to avoid mechanical wear and achieve high levels of accuracy. By blending linear and parabolic motion profiles, LSPB ensures that the robot moves in a way that balances speed, smoothness, and control, making it a versatile choice for many robotic applications.[14].

Spline Interpolation

Spline interpolation involves using piecewise polynomial functions to create trajectories that pass through multiple via points. Unlike polynomials, which are typically used to generate a trajectory between two points, splines offer more flexibility by allowing the trajectory to pass through several points in space. This is particularly useful when a robot needs to follow a complex path, such as a curve or surface.

Cubic Splines

Cubic splines are the most common type of spline used in trajectory generation. A cubic spline is defined by a piecewise cubic polynomial function that ensures continuity in position, velocity, and acceleration at each via point. Each segment of the spline is represented by a cubic polynomial. Boundary conditions ensure that the first and second derivatives (velocity and acceleration) are continuous at each via point. This results in a smooth trajectory that passes through all specified points.[15]

$$q(t)=ai+bi\ t+ci\ t^2+di\ t^3 \quad \text{for each segment I} \quad (3)$$

B – Splines

B-splines (Basis splines) are another popular choice for generating smooth trajectories. Unlike cubic splines, which are constrained to pass through all via points, B-splines allow more flexibility by approximating the path instead of strictly following every via point. This gives designers more control over the smoothness of the trajectory, as the B-spline can be adjusted by changing the control points without needing to pass through them all. B-splines are particularly useful in robotic applications where exact path-following is less important than ensuring smooth, continuous motion, such as in animation or surface inspection tasks. B-splines, or Basis splines, are flexible mathematical tools used in robotics for smooth trajectory generation, especially when passing through multiple points is not strictly required. Unlike traditional splines that pass through every via point, B-splines use control points to define the shape of the trajectory, offering more control over the smoothness and flexibility of the path. This allows for fine-tuning of the trajectory without the need for exact point-to-point adherence. B-splines are defined by piecewise polynomial functions and ensure continuity in position, velocity, and higher derivatives, which is crucial for smooth motion. [16] The ability to adjust the trajectory by moving the control points, without directly altering the path through every point, makes B-splines highly useful in applications such as path planning, animation, and complex robotic tasks, where maintaining smooth, uninterrupted motion is key. Additionally, B-splines can be scaled and adapted for various tasks, making them a versatile and robust solution for complex robotic trajectories.

Comparison Of Path Planning And Trajectory Planning

Path planning and trajectory generation are both integral components of robotic motion control, each addressing different but complementary aspects of how robots move in their environments. While they are often discussed together, they serve distinct roles and involve separate processes to ensure that a robot performs its tasks efficiently, safely, and effectively.

Path Planning (Focus On Where To Go)

Path planning is primarily concerned with determining a feasible and collision-free route for the robot to travel from its starting point to its destination. This process focuses on the spatial aspect of the robot's movement, which means that it deals with where the robot should go, rather than how it gets there in terms of time or motion characteristics. Path planning algorithms are used to generate a path that avoids obstacles and satisfies constraints such as workspace boundaries, joint limits, or task-specific requirements. In path planning, the goal is to find a valid route, often without considering the dynamics of the robot, such as velocity, acceleration, or the forces involved in moving along the path. Algorithms like A*, D*, Rapidly-exploring Random Trees (RRT), and Probabilistic Roadmaps (PRM) are commonly used in path planning.

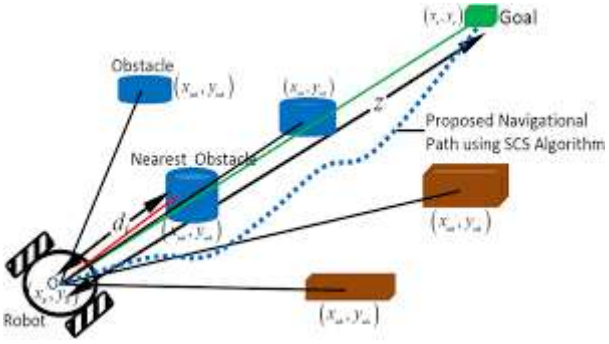

Figure 5. Representation Of Path Planning from Start to Goal point

These algorithms generate a series of waypoints or nodes in the robot's configuration space (C-space), providing a roadmap that the robot can follow to avoid obstacles and reach its destination.The key challenge in path planning is ensuring that the path is both feasible and optimal within the given constraints. For example, in an environment with static or dynamic obstacles, the path planning algorithm must compute a path that avoids collisions while minimizing factors such as distance, energy consumption, or time. In some cases, multi-criteria optimization may be necessary, where the path must satisfy several objectives simultaneously.Path planning is often used in mobile robotics, autonomous vehicles, and robotic manipulators. In mobile robots, for example, path planning helps navigate complex terrains, such as warehouse floors or outdoor environments, ensuring that the robot avoids obstacles and follows a clear, safe route.[17]

Trajectory Planning (Focus On How To Move)

Trajectory generation, in contrast, deals with the temporal aspect of the robot's motion. Once the path is determined by the path planning stage, trajectory generation specifies *how* the robot should move along that path, taking into account timing, velocity, acceleration, and dynamic constraints such as the robot's kinematic or dynamic limitations. Trajectory generation ensures that the robot moves smoothly and efficiently along the planned path. [18] It involves calculating the position, velocity, and acceleration profiles for each joint or actuator over time. This step is crucial because it defines the actual motion commands that are sent to the robot's control system, ensuring that the robot does not exceed its velocity or acceleration limits and that it moves in a controlled and coordinated manner. Common methods for trajectory generation include polynomial interpolation, cubic and quintic polynomials, splines, and Linear Segments with Parabolic Blends (LSPB). These methods are chosen based on the desired smoothness, accuracy, and dynamic constraints of the robot. For example, cubic polynomials are often used when continuous position and velocity are required, while quintic polynomials are used when smooth acceleration is also necessary. Techniques like B-splines offer even more flexibility by allowing the trajectory to be adjusted without needing to pass through every via point, making them ideal for complex or curved paths. The critical challenge in trajectory generation is balancing smoothness, speed, and accuracy while ensuring that the robot operates within its physical limitations. For example, in high-speed robotic arms, rapid acceleration and deceleration can lead to mechanical stress, so the trajectory must be carefully designed to avoid excessive wear and tear. In mobile robots, trajectory generation must account for dynamic constraints like friction, inertia, and wheel slip to ensure accurate and stable motion.[19]

Constraints In Trajectory Generation

Trajectory generation in robotics is a complex process that involves planning the precise motion of a robot over time while adhering to various constraints. These constraints are essential to ensure that the robot's movement is physically feasible, safe, and efficient. They typically arise from the robot's mechanical limitations, the environment in which it operates, and the specific tasks it is required to perform. Understanding and managing these constraints is critical for ensuring the smooth, accurate, and reliable execution of a robot's movements.

Kinematic Constraints

Kinematic constraints define the limits on a robots movement based on its geometry, joint limits, and actuator capabilities. These constraints are determined by the physical structure of the robot and include factors such as joint angles, link lengths, and the maximum and minimum positions that each joint can achieve. For example, a robotic arm may have a limited range of motion for each of its joints, which restricts the paths it can follow and the positions it can reach. Kinematic constraints also include velocity and acceleration limits.[20] Each joint or actuator of the robot can only move at a

certain maximum speed and accelerate within specific bounds. If the trajectory demands movement beyond these limits, it can result in mechanical damage or failure, so the trajectory must be designed to ensure that these limits are respected. This is especially important in industrial robots, where high-speed operations must be carefully controlled to avoid exceeding the robot's physical capabilities.

Dynamic Constraints

Dynamic constraints are related to the forces, torques, and inertial properties of the robot. They ensure that the robot's actuators can generate sufficient force or torque to follow the desired trajectory without exceeding their power capabilities. These constraints also consider the robot's mass, center of gravity, and the effects of external forces such as gravity or friction. For example, when a robotic arm moves a heavy object, the inertia of the object must be accounted for in the trajectory generation process. [21] Sudden acceleration or deceleration could cause the robot to overshoot its target or lose control due to the forces involved. Therefore, the trajectory must be generated in a way that maintains stability and control, ensuring that the forces and torques do not exceed the robot's capacity to manage them. In applications like high-speed pick-and-place robots or drones, dynamic constraints are crucial for maintaining stability during rapid movements.

Environmental Constraints

The environment in which a robot operates imposes several constraints on its trajectory. These include obstacles, workspace boundaries, and floor or terrain conditions. The trajectory must be designed to avoid collisions with obstacles, whether static or dynamic, and stay within the designated workspace. For mobile robots, environmental constraints could also include factors like terrain roughness, slopes, or friction, which affect how the robot can move. For instance, a mobile robot navigating a warehouse needs to follow a trajectory that avoids shelves, walls, and other robots, while also considering the surface it is traveling on. Similarly, a robotic arm working in a confined space must ensure that its trajectory does not cause it to collide with nearby objects or exceed its workspace boundaries. In trajectory generation, environmental constraints are typically handled by integrating obstacle avoidance techniques and ensuring that the robot remains within its safe operational zone.

Trajectory Generation Algorithms

A* Algorithm

The A* algorithm is a widely used search algorithm in robotics and computer science for pathfinding and graph traversal. It efficiently finds the shortest path between two points in a grid or graph, combining features of Dijkstra's algorithm and greedy best-first search. A* operates by maintaining a priority queue of nodes to explore, where each node is assigned a cost function f(n) based on two components: g(n), the cost from the start to the current node, and h(n), a heuristic estimate of the cost to the goal. The algorithm expands the node with the lowest f(n) value, ensuring a balance between exploring new paths and staying on course towards the goal. A* is both complete and optimal, meaning it will always find the shortest path if one exists, provided the heuristic is admissible (i.e., it does not overestimate the cost). Despite its efficiency in static environments, A* can become computationally expensive in large or dynamic spaces, as it must store all explored nodes in memory. It is particularly suited for applications such as mobile robot navigation, where a pre-mapped environment allows for effective path planning.[22]

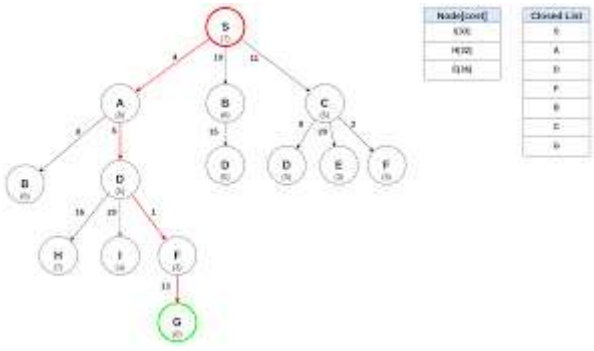

Figure 6. Representation Of A* Algorithm

D* Algorithm

The D* (Dynamic A*) algorithm is an advanced path-planning algorithm specifically designed for dynamic environments where the map or obstacles can change over time. Unlike the A* algorithm, which is suited for static environments, D* recalculates only the affected portion of the path when an obstacle is encountered, making it efficient for real-time applications. It starts by planning a path from the start to the goal based on initial information. As the robot moves and detects new obstacles or changes in the environment, D* dynamically updates the path without the need for a full replan, focusing only on the areas impacted by the changes. The robot can thus adapt quickly to new obstacles or changes in the environment. D* is widely used in mobile robots and autonomous vehicles, allowing them to navigate unpredictable or partially unknown environments with greater efficiency compared to traditional methods. Its ability to handle real-time updates makes it ideal for applications where the environment is constantly shifting or evolving.

Rapidly Exploring Random Tree (RRT)

The Rapidly-Exploring Random Trees (RRT) algorithm is a widely-used method for path planning in robotics, particularly effective in high-dimensional spaces and environments with complex obstacles. RRT builds a tree of random samples by expanding from a start position towards unexplored areas, ensuring rapid coverage of the space. At each step, the algorithm connects new random points to the nearest node in the existing tree, while checking for collision-free paths. Although RRT is fast, it does not guarantee the shortest path. To address this, RRT* (an optimized version of RRT) iteratively improves the path by rechecking connections and rewiring the tree to find shorter and more efficient paths over time. RRT* ensures asymptotic optimality, meaning that given enough iterations, it converges towards the optimal solution. Both RRT and RRT* are suitable for real-time applications and scenarios requiring flexible, scalable trajectory generation, but RRT* offers the added advantage of refining the path quality as more samples are added.[23].

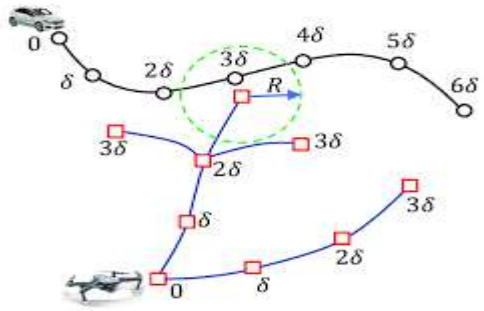

Figure 7. Representation Of RRT Method

3. RESULTS AND DISCUSSION

3.1 Applications Of Trajectory Planning

Trajectory generation plays a critical role in numerous applications across various fields of robotics and automation, providing precise control and ensuring that robots can move safely, efficiently, and smoothly. It ensures robots not only reach their destination but do so while respecting dynamic, kinematic, and environmental constraints.

3.1.2. Industrial Automation And Manufacturing

In industrial robotics, trajectory generation is fundamental to tasks such as welding, painting, assembly, and material handling. Robots in manufacturing environments must follow specific paths with precision to ensure consistency and high-quality results. For instance, in welding applications, robots need to follow predetermined paths while controlling the speed and orientation of the tool to create uniform welds. Similarly, in pick-and-place operations, robots must generate smooth trajectories to minimize the time taken to transport objects while ensuring that no obstacles are hit during the process. Trajectory generation ensures high efficiency and productivity in such automated systems.

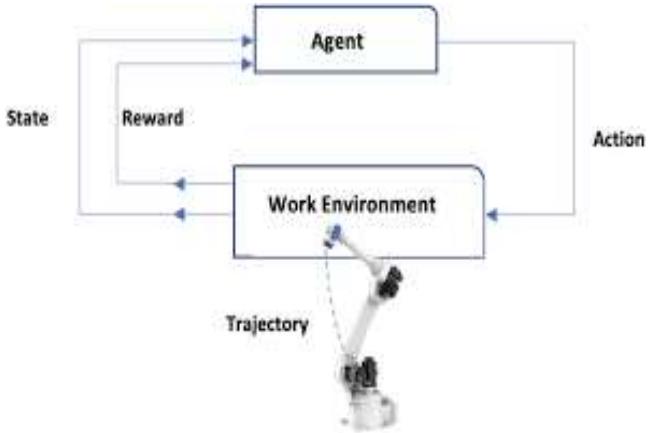

Figure 8. Trajectory generation of an pick and place manipulator

3.1.3. Autonomous Vehicles

Autonomous vehicles, including self-driving cars, rely heavily on trajectory generation for safe and efficient navigation through complex environments. These vehicles need to plan trajectories that account for obstacles (both stationary and moving), traffic regulations, and the safety of passengers. The trajectory must be generated in real-time, considering dynamic changes such as pedestrians crossing or other vehicles suddenly stopping. Trajectory

generation algorithms allow autonomous vehicles to calculate the optimal path, controlling speed and direction while avoiding collisions and adhering to traffic rules.

3.1.4. Aerospace And Drones

In aerospace applications, trajectory generation is essential for autonomous drones and spacecraft. Drones use trajectory planning to perform tasks such as surveillance, package delivery, and search and rescue missions. These unmanned aerial vehicles (UAVs) must navigate through environments with obstacles such as buildings or trees, requiring real-time adjustments to their trajectories.[24] Additionally, the trajectories must consider factors like wind, battery life, and payload constraints. In space exploration, trajectory generation is crucial for spacecraft navigation, enabling precise maneuvers when orbiting planets, docking with space stations, or landing on other celestial bodies.

3.1.5. Mobile Robot

Mobile robots, such as warehouse robots, delivery robots, and service robots, rely on trajectory generation to move from one point to another efficiently. For example, in logistics, robots need to navigate warehouses to transport goods between shelves and loading docks. They must avoid collisions with other robots, obstacles, and humans while optimizing their routes for minimal travel time. Trajectory generation helps ensure that these robots operate smoothly, following planned paths while dynamically adapting to changes in the environment.

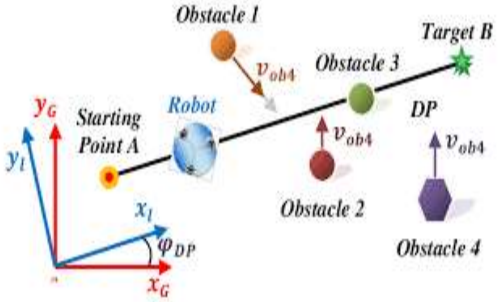

Figure 9. Trajectory generation of an mobile robot

3.1.6. Medical Robots

In medical robotics, trajectory generation is essential for robotic surgery, rehabilitation, and assistive devices. Surgical robots, such as the da Vinci system, rely on precise trajectory planning to perform minimally invasive procedures. These robots must follow delicate paths to avoid damaging tissues while providing the surgeon with enhanced precision and control. In rehabilitation, robots assisting with physical therapy use trajectory generation to guide patients through specific movements, ensuring that exercises are performed safely and effectively. Assistive robots for people with disabilities also use trajectory generation to move safely in environments like homes or hospitals.

3.1.7. Underwater Robots (ROV And AUV)

Remotely Operated Vehicles (ROVs) and Autonomous Underwater Vehicles (AUVs) rely on trajectory generation for tasks like underwater exploration, environmental monitoring, and pipeline inspection. Underwater robots must account for factors such as currents, obstacles, and pressure changes while planning their paths. Trajectory generation allows these robots to perform missions autonomously, adjusting their course as needed while ensuring energy efficiency and minimizing the risk of collisions in complex underwater environments.[25]

4. CONCLUSION

In conclusion, trajectory generation is a cornerstone of modern robotics, playing a critical role in ensuring that robots can move smoothly, safely, and efficiently through their environments. As robots become increasingly integral to various industries, from manufacturing to healthcare, aerospace, and autonomous vehicles, the need for sophisticated and reliable trajectory generation methods continues to grow. This journal explored the mathematical foundations and types of trajectories, including linear, polynomial, and spline based methods, that allow robots to move from one point to another while adhering to various constraints such as joint limits, velocity, and acceleration. Trajectory generation algorithms, such as A*, D*, RRT, and RRT*, offer versatile solutions for path planning in both static and dynamic environments. These algorithms allow robots to navigate complex spaces, avoid obstacles, and ensure real-time adaptability in response to changes. The development of these algorithms has enabled robots to operate autonomously in unpredictable environments, enhancing their applications in fields like autonomous driving, space exploration, and industrial automation. Moreover, the comparison between path planning and trajectory generation highlighted how these two concepts, though related, address different aspects of a robot's motion path planning focuses on finding a feasible route, while trajectory generation ensures the robot follows that route in a smooth and controlled manner.As the robotics field continues to evolve, future advancements in trajectory generation will likely focus on enhancing real-time capabilities,

increasing computational efficiency, and incorporating machine learning to enable more intelligent and adaptive motion planning. Overall, trajectory generation remains a fundamental element in robotic systems, bridging the gap between theoretical path planning and practical motion execution, ultimately driving forward the capabilities and potential of robotic technologies.

ACKNOWLEDGEMENTS

Special thanks to mentors and colleagues for their valuable insights and encouragement. Their support has been instrumental in shaping the ideas and content presented in this journal.

REFERENCES

[1] Z. Shiller and S. Dubowsky, "On computing time-optimal motions of robotic manipulators in the presence of obstacles," IEEE Transactions on Robotics and Automation, vol. 7, no. 6, pp. 785–797, Dec. 1991, doi: 10.1109/70.103108.

[2] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," The International Journal of Robotics Research, vol. 5, no. 1, pp. 90–98, Mar. 1986, doi: 10.1177/027836498600500106.

[3] C. R. Hargraves and S. W. Paris, "Direct trajectory optimization using nonlinear programming and collocation," Journal of Guidance, Control, and Dynamics, vol. 10, no. 4, pp. 338–342, Jul.–Aug. 1987, doi: 10.2514/3.20105.

[4] J. Craig, P. Hsu, and S. S. Sastry, "Trajectory control of robot manipulators: Robustness of the computed torque method," The International Journal of Robotics Research, vol. 9, no. 4, pp. 68–74, Aug. 1990, doi: 10.1177/027836499000900405.

[5] T. Yoshikawa, "Manipulability of robotic mechanisms," The International Journal of Robotics Research, vol. 4, no. 2, pp. 3–9, Jun. 1985, doi: 10.1177/027836498500400201.

[6] B. Siciliano, "Kinematic control of redundant robot manipulators: A tutorial," Journal of Intelligent and Robotic Systems, vol. 3, no. 3, pp. 201–212, 1990, doi: 10.1007/BF00157006.

[7] C. D. Johnson and J. E. Bobrow, "Minimum-effort motions for open-chain manipulators with task-dependent end-effector constraints," IEEE Transactions on Robotics and Automation, vol. 7, no. 3, pp. 356–360, Jun. 1991, doi: 10.1109/70.88140.

[8] W. Khalil and E. Dombre, *Modeling, Identification, and Control of Robots,* Butterworth-Heinemann, 2004, doi: 10.1016/B978-012400460-1/50007-2.

[9] N. Hogan, "Impedance control: An approach to manipulation: Part I—Theory," Journal of Dynamic Systems, Measurement, and Control, vol. 107, no. 1, pp. 1–7, Mar. 1985, doi: 10.1115/1.3140713.

[10] J. Hollerbach, "Optimum trajectory planning for robot manipulators," in *Proceedings of the 1983 International Symposium on Robotics Research,* pp. 219–227, 1983.

[11] J. Hauser, S. Sastry, and G. Meyer, "Nonlinear control design for slightly nonminimum phase systems: Application to V/STOL aircraft," Automatica, vol. 28, no. 4, pp. 665–679, Jul. 1992, doi: 10.1016/0005-1098(92)90026-Q.

[12]M. W. Spong and M. Vidyasagar, *Robot Dynamics and Control,* Wiley, 1989.

[13] T. Lozano-Perez, "Spatial planning: A configuration space approach," IEEE Transactions on Computers, vol. 32, no. 2, pp. 108–120, Feb. 1983, doi: 10.1109/TC.1983.1676196.

[14] A. A. Maciejewski and C. A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," The International Journal of Robotics Research, vol. 4, no. 3, pp. 109–117, 1985, doi: 10.1177/027836498500400306.

[15] J. Latombe, *Robot Motion Planning,* Springer, 1991, doi: 10.1007/978-1-4615-4022-9.

[16] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation,* CRC Press, 1994, doi: 10.1201/9781315136370.

[17] K. G. Shin and N. D. McKay, "Minimum-time control of robotic manipulators with geometric path constraints," IEEE Transactions on Automatic Control, vol. 30, no. 6, pp. 531–541, Jun. 1985, doi: 10.1109/TAC.1985.1103987.

[18] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB,* Springer, 2011, doi: 10.1007/978-3-642-20144-8.

[19]J. E. Bobrow, S. Dubowsky, and J. S. Gibson, "Time-optimal control of robotic manipulators along specified paths," The International Journal of Robotics Research, vol. 4, no. 3, pp. 3–17, Sep. 1985, doi: 10.1177/027836498500400301.

[20] A. De Luca and G. Oriolo, "Trajectory planning and control for planar robots with elastic joints," IEEE Transactions on Robotics and Automation, vol. 19, no. 2, pp. 146–156, Apr. 2003, doi: 10.1109/TRA.2003.808865.

[21] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings of the 2000 IEEE International Conference on Robotics and Automation,* pp. 995–1001, 2000, doi: 10.1109/ROBOT.2000.844730.

[22]D. E. Koditschek, “Robot planning and control via potential functions,” Robotics Review, vol. 2, pp. 349–367, 1992.

[23] M. T. Mason, “Compliance and force control for computer-controlled manipulators,” IEEE Transactions on Systems, Man, and Cybernetics, vol. 11, no. 6, pp. 418–432, Jun. 1981, doi: 10.1109/TSMC.1981.4308751.

[24] K. L. Doty, C. Melchiorri, and C. Bonivento, “A theory of generalized inverses applied to robotics,” The International Journal of Robotics Research, vol. 12, no. 1, pp. 1–19, 1993, doi: 10.1177/027836499301200101.

[25] A. Bicchi and G. Tonietti, “Fast and soft-arm tactics: Dealing with the safety-performance trade-off in robot arms design and control,” IEEE Robotics & Automation Magazine, vol. 11, no. 2, pp. 22–33, Jun. 2004, doi: 10.1109/MRA.2004.1310949.