# Formal Modeling and Verification of Industrial Robotic Arm - A Case Study

Ilmaan Zia[1], Muhammad Abdul Basit Ur Rahim[1,*], Tairan Liu[2] and Zhangying He[1],
[1]Department of Computer Science and Engineering, California State University Long Beach, CA, USA
[2]Department of Mechanical and Aerospace Engineering, California State University Long Beach, CA, USA
ilmaan.zia01@student.csulb.edu, m.basit@csulb.edu, tairan.liu@csulb.edu, mandy.he@student.csulb.edu
*Muhammad Abdul Basit Ur Rahim

*Abstract*—Robotic arm systems are revolutionizing industries by automating operations and extending capabilities beyond human limitations, prioritizing accuracy and precision. These systems redefine boundaries across various sectors, including healthcare, mobile automation, and space exploration. Ensuring the safety of the robotic arm and its surrounding environment is paramount during its operations. This paper presents a detailed investigation of formal modeling and verification in the context of an industrial robotic arm, specifically a multi-joint mechanism equipped with advanced capabilities such as object manipulation, obstacle detection, and avoidance. The study uses the model-checking tool to capture the complex behavior of the system precisely. In-depth simulation and verification techniques are used for the formal model, focusing on its crucial properties, including safety, liveness, fairness, and deadlock-freeness.

*Keywords–Robotic arm, Formal verification, Model-checking, Safety, Fairness, Liveness, Deadlock-freeness, Self-adaptive system.*

## 1. Introduction

Automation has become increasingly vital in various industries. With their versatility and precision, robotic manipulators offer significant potential for streamlining operations. Assisted by camera systems and computer vision, robot manipulators can obtain the capabilities of real-time object detection, collision avoidance, and precision manipulation. These capabilities prepare robotic manipulation systems with the potential to revolutionize various industries.

A robotic manipulation system utilizes camera systems and object detection algorithms to identify and grasp objects within its workspace. Ensuring the safety of the robotic arm and its surroundings is paramount. The system must be self-managing and adaptable to achieve objectives [1], handle runtime errors [2], and reconfigure correctly [3]. However, designing such self-adaptive systems is challenging due to their complexity and dynamism. Using formal verification early in the design process helps identify flaws and assure operational correctness [4]. This study incorporates formal techniques to ensure the robotic arm's reliability in detecting and manipulating objects. To address these concerns, this study employs formal modeling and verification techniques to comprehensively analyze the robotic arm's behavior and assess various possible outcomes. The formal verification process involves systematically examining the system's performance, identifying potential failure scenarios, predicting outcomes that may pose risks, capturing the behavior of the object detection algorithms, as well as the coordinated movements of the robotic arm's joints and gripper. Verification is performed to assess the system's ability to accurately detect objects, execute smooth and precise manipulations, and safely transport them to desired locations. Through formal verification, the performance and safety of the robotic arm can be rigorously assessed, ensuring compliance with desired specifications and industry standards. By examining all possible failure scenarios, engineers gain insights into critical failure points, enabling them to implement proactive measures to prevent harm to the robot or its surrounding environment.

The practical implications of this research are significant for various industries. Deploying such robotic arms in manufacturing facilities, warehouses, and logistics centers can enhance productivity, reduce human error, and optimize workflow efficiency. Industries can benefit from streamlined material handling processes, improved inventory management, and reduced labor costs.

The research focuses on an industrial robotic arm equipped with object identification, obstacle detection, and path optimization capabilities. It undergoes formal verification and behavioral analysis using UPPAAL for correctness, safety, liveness, fairness, and deadlock-freeness. Subsequently, the system is developed and tested for various functions, including object handling and path optimization.

The outline of the paper is as follows. We present related work in section 2. Section 3 briefly describe the basics. In section 4, the methodology is proposed. In section 5, we briefly present the robotic arm system of study. In section 6, we describe a mechanical system to manipulate within its environment. In section 7, we introduce obstacle detection and avoidance. In section 8, describes the formal verification of the robotic arm. Sec. 9 concluded the research.

## 2. Related Work

Automation systems must be self-adaptive which usually contain internal components of feedback, analyzer, and decision-maker for managing self-adaptations and reconfigurations. Common concerns that arose are uncertainties at runtime and thus a way of managing the uncertainties and assuring the correctness of adaptations is crucial. In this section, we first present some challenges of self-adaptive systems from existing works. Next, we review the common formal methods of specifying self-adaptive systems, followed by a review of

verification methods that can ensure and guarantee correctness with defined properties and constraints.

### 2.1 Challenges of Self-Adaptive Systems

One key component of self-adaptive systems is their ability to handle changes during runtime. It is a key difference between self-adaptive systems and other systems. In this context, self-adaptive systems must consider both offline activities during design and online activities during runtime. As a result, finishing the design only counts as finishing half of the work [5]. During runtime operations, the systems face various sources of uncertainties coming from the external environment or internal components. So, self-adaptive systems must consider, design, and implement the adaptive configuration logic to deal with the various scenarios of operational activities [6], and they must be done appropriately and robustly so that the systems can adapt to the changing environment, partial failures, and errors during operations.

A key question to answer for designing a self-adaptive system is how can we assure the correctness of the self-adaptability [4]. Many existing works have suggested that providing formal specifications and verification in the early design is an appropriate technique for achieving this fundamental goal. However, among the existing works for autonomous robotic systems, only around one-third of the prior works use formal specification, modeling, or validation tools to check the correctness [7]. The reason for the lack of using formal methods, according to [7], is primarily due to a lack of methods and tools available for conducting formal methods. Often, models or specifications of similar components are incompatible and locked into a particular tool. Each tool is only suited to the modeled component or the properties of interest. However, a common aspect is to use s formal method to tackle these challenges [8].

### 2.2 Formal Specifications and Verifications

Luckcuck *et al.* [7] reviewed hundreds of papers that employ various methods to specify and verify autonomous robotic systems which require self-adaptivity. We highlight the most used methods here. According to their survey, formal methods usually specify a system's behaviors and properties. Regarding behavior specification, set-based formalisms are the most common approach, such as Weys *et al.* [9] uses approaches of Z-model, Liang *et al.* [10] uses Java animator for Z specifications, and Tarasyuk *et al.* [11] uses Event-B specifications to describe the system's behaviors. Set-based methods use set-theoretic representation and manipulation of data. Their advantages reside in the suitability to capture data structures of the described system, but they have the limitation of capturing a full range of system behaviors, particularly uncertain runtime behaviors.

The second largest method used in formulism is logics, ranging from temporal logic [12], [13], probabilistic temporal logic [14], [15], to dynamic logic [16], [17]. In this method, they usually synthesize a system's behaviors in a model checkers tool such as UPPAAL or PRISM, then specify its properties and constraints in the tool to verify whether satisfied or not.

Another approach is using State-Transition Formalisms such as Petri Nets and Finite-State Automata (FSA) to specify the behaviors of a state-transition system which often captures time and probabilistic transitions. Some authors combine temporal logic and FSA [18], and others use Petri Nets to capture the abstract architecture of the robotic agents [19]–[21] and Iftekhar *et al.* [22] use The UPPAAAL model checker with timed automata to describe input models and verify against temporal logic properties. [23] uses each Markov chain to describe the probability for an individual robot to be in a specific state in a Finite State Transition Diagram. The robot swarm aggregation patterns are modeled in Markov chains and then simulated in Webots simulator [24] to test the model.

The most popular approaches are using a combination of state transition formalism and checking in a model checker tool. Other approaches use Communicating Sequential Process (CSP) [25] and ontologies [26], [27] to specify the system. Mathematical models such as process algebra Finite State Processes (FSPs) are used to define safety and liveness properties, followed by architecture checking written in $\pi$ calculus combined with ADL ($\pi$ADL) [28] to verify if the design can fulfill the desired properties.

## 3. Background

**Model Checking** Model checking is an automatic technique for verifying finite state systems [29]. The Specifications are expressed in temporal logic, and the system is modeled as a state transition diagram. An efficient search procedure is used to determine whether or not the state transition model satisfies the specifications.

**Timed Automata:** In UPPAAL, the system is modeled using timed automata, similar to the finite state machine (FSM) diagram of a unified modeling language (UML). The timed automata are more formal as compared to FSM. It consists of states, transactions, guards, channels, and assignments. Each system component is modeled separately, and the components communicate using the channels. The channels can be prioritized over other channels [30][31].

**Temporal logic** is any system of rules and symbolism for representing and reasoning about propositions qualified in terms of Time [32]. For specification, UPPAAL supports timed computation tree logic (TCTL), an extension of computational tree logic (CTL).

The liveness describes that system is progressing to achieve a specific goal [33], and fairness is stated as (PUQ) where q is bound to happen, and until it happens, p will hold [34]. Moreover, the safety property ensures that the system is safe to use.

## 4. Proposed Methodology

Figure 1 represents the proposed methodology. The robotic arm is formally modeled using timed automata, and constraints are defined using temporal logic. For formal verification, we have used the UPPAAL model checking tool, which results in

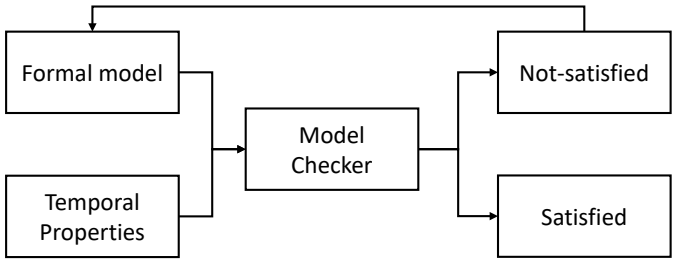


Figure 1. Proposed methodology

whether the model satisfies the properties or not. The model checking tool generates the counterexample if the property doesn't satisfy. Then the design engineer revises the model and reiterates the verification process. We have implemented the system after formally verifying the model.

## 5. Robotic Arm - Case Study

We have modeled a robotic arm capable of picking objects from one location and dropping them at the desired location. The system comprises three joints and a gripper to pick and drop objects. The joints work in coordination with each other with the help of an encoder which measures the angle for each joint. The controller coordinates between the encoder and the arms. Moreover, the camera identifies the object's size and shape and adjusts the gripper accordingly to the required orientation. This study aims to analyze the robotic arms working and behavior in real-life scenarios. It will ensure the system will not harm itself or the surroundings and how it can detect and avoid obstacles.

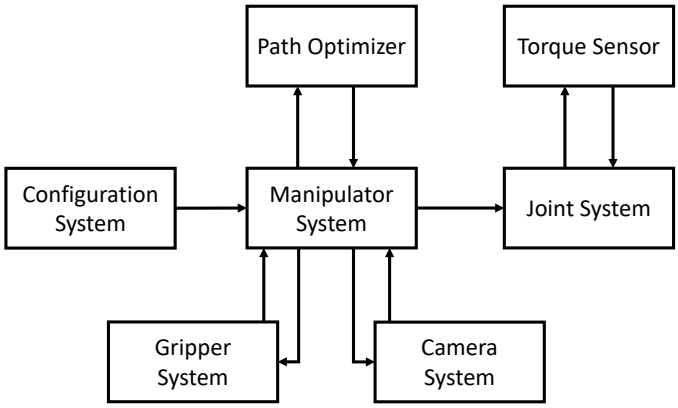


Figure 2. Architecture of robotic arm

Figure 2 depicts the comprehensive architecture of the robotic arm, showcasing the interconnections among its various components. All components seamlessly integrate with the manipulator system. Notably, the configuration system follows a unidirectional data flow pattern, while the path optimizer, joint system, camera system, and gripper system exhibit bidirectional data exchange. Furthermore, the torque sensor establishes connections with the joint and manipulator systems, effectively contributing to the arm's overall functionality.

### 5.1 Robotic Manipulator System

The robotic manipulator system consists of three interconnected links connected to the preceding link. The first arm is attached to the base, and the second arm forms the initial segment of the arm assembly. The second arm is connected to the first and third, serving as the intermediate section. Finally, the third arm is linked to the second arm and houses the gripper, which allows for object grasping and release. The purpose of the arm system is to manipulate objects in a controlled manner.

The arm system receives signals from encoders that relay the desired angles for each joint. The controller processes these signals and calculates the appropriate angles to position the arm system accurately. By adjusting the angles of each joint, the arm can be directed to reach specific objects, facilitating precise object manipulation tasks.

The manipulator system transitions through various states to grasp and transport objects, as illustrated in Figure 3. It starts in an initial readiness state, awaits source position inputs, engages path planning, and then enters a movement state operated by the Joint System. The system recalibrates paths to avoid detected obstacles. Upon reaching the target object, it transitions to grab the object, returning to path planning for the drop location while avoiding hindrances. At the drop spot, it releases the object and returns to the initial position. This synchronized sequence demonstrates the arm's agility and adaptability in dynamic environments

### 5.2 Joint System

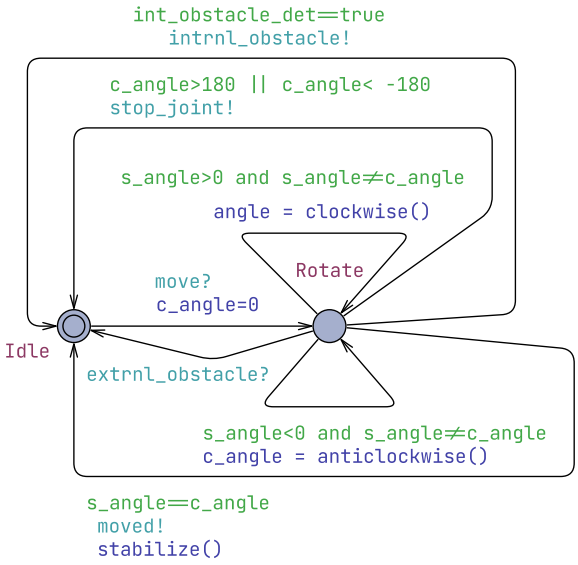


Figure 4. Timed automata of joint system

The robotic arm contains three joints that connect the arm segments and enable articulated motion. The joints set the angle of each arm segment, determining overall arm position and orientation. A controller and encoder system precisely controls the joint angles. The controller adjusts the angles to

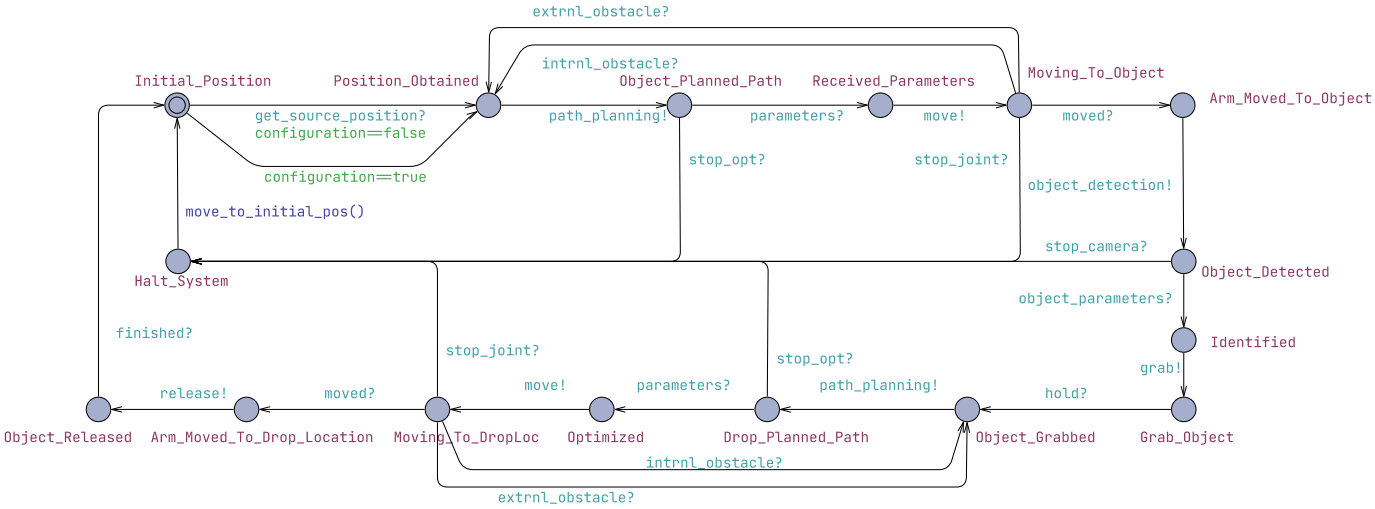


Figure 3. Timed automata of manipulator system

reach objects at various heights and provide flexibility through motions like rotation, bending, and extension. This allows enhanced dexterity and versatility when manipulating objects. A planar arm with three revolute joints has a large dexterous workspace in the plane (given "good" link lengths and large joint ranges), because any position in the interior of its workspace can be reached with any orientation.

The joint angle range from -$\pi$ to +$\pi$ provides flexibility and adaptability to the robot arm system. It allows for smooth and continuous movement, enabling the arm to navigate around obstacles and reach objects from various angles. This range facilitates precise positioning and manipulation, enhancing the arm's capability to perform complex tasks precisely and accurately.

As depicted in figure 4, the joint system idles until receiving a move signal. It then rotates the arm clockwise or counter-clockwise to the desired angle. Upon reaching the target angle, the joint stabilizes to stop further rotation and signals the arm system. If the angle exceeds predefined joint limits of -$\pi$ to +$\pi$ during rotation, it triggers a stop signal to halt arm movement."

### 5.3 Gripper System

The gripper is seamlessly integrated with the third link of the robotic system. The primary responsibility of the gripper is to intelligently identify the object's size, orientation, and shape, dynamically adjusting its configuration to ensure successful object manipulation. This task is accomplishedusing by a cutting-edge camera system installed within the gripperf. The camera enables real-time object recognition by leveraging sophisticated imaging technology, empowering the gripper to accurately assess the targeted object's size, shape, and orientation. The gripper provides an adjustable range of configurations, enabling it to accommodate diverse object shapes and sizes easily. Moreover, the gripper handles precisely transporting objects from one location to another, ensuring seamless and efficient movement. This case study serves as a testament to the remarkable capabilities of the camera-enabled gripper within the robotic arm system, showcasing its remarkable object recognition, adaptability, and transportation.

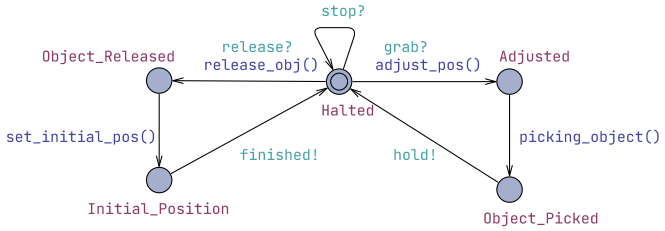


Figure 5. Timed automata of Gripper system

Figure 5 illustrates the flow of the gripper system, Initially the gripper system is in Halted state, waiting for the grab signal from the arm system; once the grab signal is received, the system adjusts the gripper position to pick up the object and goes to Adjusted state. Once the gripper picks the object, it goes to the Object Picked state and sends a hold signal to the arm system. After successfully grabbing the object and transferring it to the drop location, the arm system sends a release signal to the gripper; the gripper releases the object, goes to the Object Released state, and sets the gripper orientation to the initial position. After the successful cycle, the gripper system sends a finished signal to the robotic arm.

### 5.4 Camera System

The camera component integrated with the gripper system plays a pivotal role in the overall functionality of the robotic arm. It is designed to capture high-resolution images of the objects within its field of view, enabling precise object recognition. Equipped with advanced image processing algorithms, the camera swiftly analyzes the captured images to identify important object characteristics such as size, shape,

and orientation. This critical information is the basis for the gripper's subsequent adjustments to ensure optimal object handling. The camera's real-time capabilities allow for rapid and accurate object identification, making it an invaluable asset in enhancing the gripper's overall performance. Through the seamless integration of a camera within the gripper, the robotic arm system gains the ability to perceive and interpret the surrounding environment, facilitating intelligent decision-making for efficient and reliable object manipulation.

Apart from its function in object manipulation, the camera also excels in obstacle detection. Through keen observations and analytical capabilities, it identifies potential obstructions, enabling the robotic arm to anticipate and avoid obstacles. This harmonious integration of image processing and obstacle detection within the camera equips the robotic arm with the ability to understand its environment and make well-informed decisions for navigating and manipulating objects efficiently, effectively, and with heightened safety.

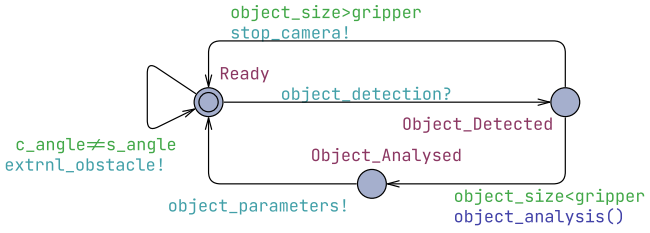


Figure 6. Timed automata of camera system

Initially, the camera system is in a Ready state. Once the arm system sends an object detection signal, the camera system analyzes the object size and orientation and checks if the object size is greater than the gripper size. On successful analysis, the system sends object parameters to the arm system for further operation. If the object size exceeds the gripper size, the camera system sends a stop signal to the arm system to stop further operations.

Moreover, in its readiness, the camera system exhibits a proactive role. While the arm system is yet to attain its final destination, the camera system vigilantly scans the surroundings for potential external obstacles. As the camera system detects any external obstacles, it instantaneously triggers an obstacle detection signal, alerting the manipulator system for possible obstructions.

### 5.5 Path Optimization System

A path optimizer is a crucial component in a robotic arm system that helps optimize the trajectory or path the arm takes to perform a specific task. The main goal of a path optimizer is to determine the optimal path (in terms of distance, time, energy consumption, etc.) for the robot arm to move from its current position to the desired position while considering various constraints and objectives.

A path optimizer considers factors such as constrained environments, obstacles in the environment, joint angle limitations, joint angular speed limit, and energy efficiency to generate an optimized path. By analyzing these factors, the optimizer can find a path that avoids collisions with obstacles, respects the joint angle constraints, and minimizes the overall energy consumption of the robot arm.

The path optimizer employs algorithms like motion planning and optimization techniques to consider robot arm constraints and environmental factors. It assesses various paths based on distance, time, energy, and motion smoothness, selecting the most suitable path. The optimized path consists of waypoints or joint configurations for precise positioning.

The robotic arm can perform tasks more efficiently by utilizing a path optimizer, reducing unnecessary movements, minimizing energy consumption, and avoiding collisions with obstacles. This optimization improves the overall performance and productivity of the robotic arm system, allowing it to operate effectively in complex unstructured, and dynamic environments.

The path optimization system receives a signal from the manipulation system to perform path planning based on the constraints. The calculated joint angles will maintain inside the permissible range of constraints (e.g., maximum and minimum angles, $\pi$ and $-\pi$, respectively). Once a feasible and optimal path is obtained, the path will be forwarded to the manipulation system. If the angles calculated don't satisfy the constraints (greater than $\pi$ or less than $-\pi$), the system sends a stop signal to the arm system, resulting in a stop to the robotic arm system. This process is illustrated in Figure 7.

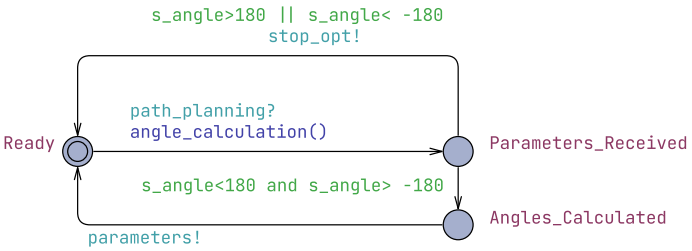


Figure 7. Timed automata of path optimizer system

### 5.6 Torque Sensor

Integrated within the joint system, the Torque Sensor is a critical component within the robotic framework. Its primary task revolves around detecting variations in torque. The torque sensor remains in constant communication with the joint, adeptly identifying any unusual changes in movement. This ability to perceive even the subtlest shifts in torque enables the sensor to play a key role in obstacle detection.


Figure 8. Timed automata torque sensor

Operating initially in an idle state, the sensor patiently awaits its moment of action. It undergoes a transformative shift when detecting any anomalous movement indicative of an obstacle. It updates the value of an internal variable known as “internal obstacle detection” to the truth state. Further, it transmits the signal to the manipulator system, conveying the presence of an obstacle.

### 5.7 Configuration System

The Configuration System acts as an intermediary between the end user or other subsystems and the main control system of the robotic arm. Its primary roles include obtaining the source position of the target object and forwarding the object and drop positions to the main system for execution. Initially, the Configuration System is in the Idle state, waiting for input to obtain the source position. When the system is triggered, the Configuration System sets the source position of the target object.

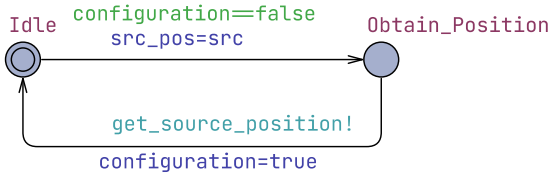


Figure 9. Timed automata of configuration system

The Configuration System is vital for the Robotic Arm’s operation. It receives the object’s source position from users or subsystems, then transitions to ”Obtain Position.” It signals the central control system with ”get source position” to indicate data readiness. The central control system plans and executes robot actions, including object manipulation with the arm’s end-effector and gripper.

The Configuration System sets the volume of the workspace that the manipulator’s end-effector can reach. There are two types of workspaces

- Dexterous workspace is the volume of space that the robot end-effector can reach with all orientations. At each point in the dexterous workspace, the end-effector can be arbitrarily oriented.
- The reachable workspace is the volume of space that the robot can reach in at least one orientation.

## 6. Manipulator

Refers to a mechanical system or device that is designed to manipulate or control objects within its environment. It typically consists of a series of rigid links connected by joints, with an end effector (such as a gripper or tool) attached to the final link.

### 6.1 Solvable Manipulator

A manipulator is considered “solvable” if its inverse kinematics problem can be solved analytically or through closed-form solutions. A manipulator will be considered solvable if the joint variables can be determined by an algorithm that allows one to determine all the sets of joint variables associated with a given position and orientation. The solvable manipulator has several advantages:

- Efficiency: Analytical solutions for inverse kinematics can be computationally efficient since they provide direct formulas for calculating the joint angles.
- Accuracy: Solvable manipulators can provide precise control over the robot’s motion since the joint angles are calculated exactly based on the desired end effector position and orientation.
- Predictability: With analytical solutions, the behavior of the manipulator is deterministic and predictable.
- Simplified Programming: Solvable manipulators often have straightforward programming interfaces, as the inverse kinematics equations can be directly incorporated into the control software

## 7. Obstacle Detection And Avoidance

Obstacle detection and avoidance for robot manipulators are critical aspects of ensuring safe and efficient operation in various industrial applications. In this work, we employ a camera-based solution to detect potential collision situations for the robotic manipulator. Cameras are installed in the environment from several different directions facing the operation space to provide visual feedback from various perspectives. All the images collected from the cameras are processed by a computer vision program running on the computer. The obstacles in the environment are identified for the path-planning process.

The mechanism for collision avoidance in this work is based on the multiple-solution feature of inverse kinematics equations. For a robotic manipulator, inverse kinematics is used to determine the values for the robot joint variables given the position and orientation of the end-effector.

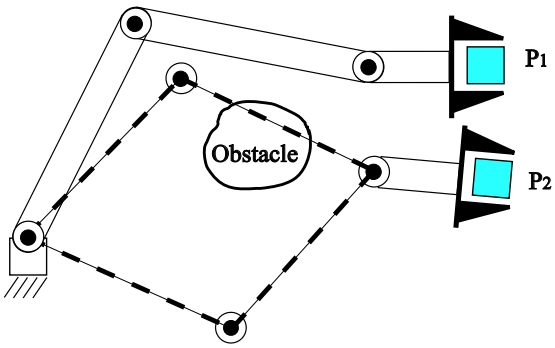


Figure 10. Obstacle avoidance is needed when the manipulator is grabbing an object at $P_2$.

Due to the fact that the inverse kinematic equations may have multiple solutions in many configurations, the manipulator has to choose one solution out of them. The most reasonable choice would be the closest solution to the previous state in an

open space without obstacles. However, another solution will be needed to avoid collisions in the presence of obstacles or in a constrained space. This situation is illustrated in Figure 10. When the robotic manipulator is commanded to move an object from $P_1$ to $P_2$ with the prescribed orientations, the easiest solution is to rotate the first link for a small angle (the dashed lines at the top) if the obstacle doesn't exist. However, with the presence of the obstacle, this solution is no longer feasible. Luckily, we have another solution, which mirrors the first two links (the dashed line at the bottom). In this way, the manipulator could deliver the object to the desired location and orientation without being interfered with by the obstacle.

## 8. Property Verification

The properties of the robotic arm system verify that the arm works correctly. This section describes the variables, constants, and constraints of the system that we have set for system verification.

In a workspace, when additional static barriers are present, the robot arm needs to be controlled activates. The arm has a predetermined starting configuration and uses the final configuration from each movement as the initial configuration for the subsequent assignment. The component tries to avoid obstacles while attempting to accomplish the tasks. The following presumptions and constraints are made in addition to this scenario information.

- Assumption 1: The arm turns on in an unidentified workplace environment. Using a mathematical technique, collision detection is accomplished assuming each link has proximity sensors installed.
- Assumption 2: The joint angle limits are considered ($-\pi$, $+\pi$).
- Assumption 3: The algorithm is solely concerned with achieving the goal and not with the end effector's orientation in any particular way.
- Assumption 4: The distance between the end effector and the target, as well as the distance between each obstacle and each link, are known at every time instant.
- Assumption 5: Multiple solutions are defined, and if an obstacle is detected, the manipulator picks the next best solution to reach the object.
- Assumption 6: The static barriers in the workspace are stationary and do not move or change their positions during the robot's operation.
- Assumption 7: The workspace is free of dynamic obstacles or moving entities that could interfere with the arm's movements.
- Assumption 8: Considering the defined constraints, the arm's path optimizer algorithm can generate optimal or near-optimal paths for reaching the object and the drop location.
- Assumption 9: The arm's gripper can securely grip and release objects of various shapes and sizes without any slippage or instability.
- Assumption 10: The obstacle detection sensors provide accurate and reliable information about the presence and position of obstacles, enabling the arm to avoid collisions effectively.

### 8.1 State Variables

The following state variables define the dynamic aspects of the system, capturing the positions, orientations, and interactions of the robotic arm and its components:

- $\theta_1$, $\theta_2$, $\theta_3$: Joint angles for the three arm segments.
- X, Y, Z: Cartesian coordinates representing the position of the end effector in 3D space.
- GripStatus: A binary variable representing the gripper's releasing (0) or grabbing (1) state.
- ObjectLocation: A tuple $(X_o, Y_o, Z_o)$ representing the coordinates of the object to be picked up.

### 8.2 Constants

The following constants define key variables that affect how the robotic arm functions and interacts, governing the system's behavior and capabilities.

- $\theta_{\text{min}} = -180°$, $\theta_{\text{max}} = 180°$: Joint angle limits for each arm segment.
- `MaxArmCapacity`: A positive real number representing the maximum weight the gripper can handle.
- `MinObstacleDistance`: A positive real number representing the minimum distance the robot should maintain from obstacles during movement.
- `MaxEndEffectorDistance`: A positive real number representing the maximum distance the gripper can reach for successful grasping.

### 8.3 Constraints

- Joint angle limits prevent overextension and ensure safe operation: $-180° \leq \theta_1 \leq 180°$, $-180° \leq \theta_2 \leq 180°$, $-180° \leq \theta_3 \leq 180°$: .
- Workspace boundaries restrict the robot's movements within a specified region: $X_{\text{min}} \leq X \leq X_{\text{max}}$, $Y_{\text{min}} \leq Y \leq Y_{\text{max}}$, $Z_{\text{min}} \leq Z \leq Z_{\text{max}}$.
- Minimum distance to avoid collisions with obstacles during movement: $||(X - X_o, Y - Y_o, Z - Z_o)|| \geq$ MinObstacleDistance.
- The gripper's state is either grabbing (1) or released (0) and must be controlled accordingly during object manipulation: GripStatus $\in \{0, 1\}$.
- The distance between the end effector and the target object must be within the gripper's range for successful grasping: $||(X - X_o, Y - Y_o, Z - Z_o)|| \leq$ MaxGripperDistance.
- The robotic arm's gripper can only handle objects with a weight that does not exceed its capacity: ObjectWeight $\leq$ MaxGripperCapacity: .

### 8.4 Temporal Properties

UPPAAL uses temporal logic operators under TCTL to represent verifiable properties. These operators are represented within UPPAAL somewhat dissimilar to usual temporal operators and will be outlined in the following paragraphs. As with standard CTL, in UPPAAL, most temporal operators

consist of two parts: a prefix and a suffix. The prefix describes how the operator checks paths from a state, while the suffix describes how the operator checks the state along each path. When verifying properties, we can place an operator before an expression to evaluate it for the entire model. The prefixes `A` means all paths and `E` means there exists a path. The suffixes `[]` means all states in a path, and `<>` means some state in a path. When combined, the full operator describes what is necessary for the verified property to evaluate as true for a state. We can also use an additional temporal operator in UPPAAL, which does not fit the earlier definition. This is the leads-to operator, represented with `-->`. Unlike the rest of the temporal operators, it is binary, meaning it operates on two expressions, not one. It can also be defined using the other operators, like so: `A[] (p imply A<> q)`, where `imply` is the logical implication operator.

Temporal operators are used to specify the temporal logic. The following temporal expressions are described as `A[] p`: where the property `p` must be true for all states in all paths (invariably); `A<> p`: where `p` must be true for some state in all paths (inevitable); `E[] p`: where `p` must be true for all states in some path (potentially always); `E<> p`: where `p` must be true for some state in some path (reachable); `p --> q`: for all states where `p` is true, the expression `q` must be true for some state in all paths from that state (leads-to).

We have verified the robotic arm's safety, liveness, fairness, and deadlock-freeness. The robotic arm and its component are verified against the following temporal properties

#### 8.4.1 Safety property

- **S1** The safety property of the robotic arm includes preventing the overextension of the arm beyond its physical limitations.
- **S2** The robotic arm must avoid collisions with obstacles when reaching, grasping, and moving objects, operating safely without harming itself, the objects, or the environment. If an obstacle is detected, the arm must take evasive actions to navigate around it.
- **S3** The robot's movement is restricted within a defined workspace, which is often determined by the dimensions and reach of the robot. The robot must be operated within these boundaries to avoid collisions with objects or the environment.

#### 8.4.2 Liveness property

The robotic arm's liveness property ensures it can reach its intended goals and complete its tasks successfully.

- **L1** The component should be able to move to the specified object location and grasp the object. After grasping the object, the arm should successfully plan and execute a path to the drop location and release the object.
- **L2** The liveness property also includes continuously accepting input for object locations and executing the required movements without getting stuck or halting indefinitely.

#### 8.4.3 Fairness property

The fairness property for the robotic arm ensures that all objects and drop locations have a fair chance to be processed without any particular object or location being prioritized consistently. Fairness also includes ensuring that the arm's response to object location inputs is prompt and unbiased, allowing each input to be processed promptly.

- **F1** The arm's path optimizer should provide an equitable distribution of movement plans for different objects and drop locations.
- **F2** The robotic arms fully extend and retract when required.
- **F3** The gripper returns to the initial position after releasing the object.

#### 8.4.4 Deadlock-freeness

Deadlock freeness property ensures that the robotic arm does not get stuck in a state where it cannot make any progress or continue its operations. The component should avoid getting into situations where it cannot move due to conflicting paths or obstacles that cannot be bypassed.

- **D1** If an obstacle is detected while moving towards an object or the drop location, the arm should dynamically adjust its path to find an alternative route and avoid deadlock scenarios.

The system satisfies all the above-stated temporal properties. Later, we simulated the robot in the defined workspace. Through this case study, we aim to gain insights into the robot's capabilities, analyze its kinematics, and evaluate its performance in various scenarios. Such modeling and analysis enable us to optimize the robot's behavior and make informed decisions for real-world applications.

#### 8.4.5 Results

In the results section, we conducted a comprehensive verification process using the UPPAAL model checker to ensure the validity of the designed system properties. Our analysis encompassed various critical aspects, including safety, liveness, fairness, and deadlock-free properties. By subjecting the system to rigorous scrutiny, we were able to confirm its compliance with these essential requirements.

To quantify our findings, we systematically recorded execution time and memory consumption for each property verification. The obtained data, presented in Table I and II, sheds light on the efficiency and resource utilization of the verification process. This analysis not only highlights the robustness of our system but also provides valuable insights into its performance characteristics.

## 9. Conclusion

The formal modeling, verification, and simulation of a complex robotic arm equipped with abilities of object manipulation, obstacle detection, and avoidance were all included in our work, which was completed successfully. We explored the details of our design throughout the process, demonstrating its ability to produce a system that is both functional and meets the requirements for that system. We revealed the

TABLE I
PROPERTY VERIFICATION

| **Property** | **Temporal Property** |
|---|---|
| Safety **S1** | A[] ((Path Optimizer.Ready && s_angle<180) imply Path_Optimizer.Angles_Calculated) |
| Safety **S2** | E[] ((Joint System.Rotate && internal obstacle detection==true) imply Joint System.Idle) |
| Liveness **L1** | E<> (Arm System.Initial Position imply Arm System.Object Released) |
| Liveness **L2** | E[] (Arm System.Initial Position imply Arm System.Initial Position) |
| Fairness **F2** | E<> ((Joint System.Idle && s_angle==180 \|\| s_angle==0) imply Arm System.Arm Moved To Object) |
| Fairness **F3** | E[] (Grabber System.Adjusted imply Grabber System.Initial Position) |
| Deadlock-freeness **D1** | A[] not deadlock |

TABLE II
TIME AND MEMORY CONSUMPTION

| **Property** | **Execution Time(seconds)** | **Memory Consumption(MBs)** |
|---|---|---|
| Safety **S1** | 0.005 | 62.928 |
| Safety **S2** | 0.002 | 52.52 |
| Liveness **L1** | 0.002 | 62.928 |
| Liveness **L2** | 0.016 | 63.01 |
| Fairness **F2** | 0.002 | 55.41 |
| Fairness **F3** | 0.001 | 55.43 |
| Deadlock-freeness **D1** | 0.004 | 62.42 |

system design's behavior across a range of circumstances by rigorously analyzing it against a variety of functional and non-functional benchmarks. With the help of our model and simulation, our study greatly enables designers and engineers working in the field of robotic arms by giving them essential insights and techniques to comprehend and carefully build the system. In essence, our research creates a link between theory and practice that encourages creativity and brings in a new era of accuracy and efficiency in the field of developing robotic arms.

## REFERENCES


[1] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003. DOI: 10.1109/MC.2003.1160055.

[2] P. Oreizy, N. Medvidovic, and R. Taylor, "Architecture-based runtime software evolution," in *Proceedings of the 20th International Conference on Software Engineering*, 1998, pp. 177–186. DOI: 10.1109/ICSE.1998.671114.

[3] B. H. C. Cheng, K. I. Eder, M. Gogolla, *et al.*, "Using models at runtime to address assurance for self-adaptive systems," in *Models@run.time: Foundations, Applications, and Roadmaps*, N. Bencomo, R. France, B. H. C. Cheng, and U. Aßmann, Eds. Cham: Springer International Publishing, 2014, pp. 101–136. DOI: 10.1007/978-3-319-08915-7_4.

[4] M. Hachicha, R. B. Halima, and A. H. Kacem, "Formal verification approaches of self-adaptive systems: A survey," *Procedia Computer Science*, vol. 159, pp. 1853–1862, 2019. DOI: https://doi.org/10.1016/j.procs.2019.09.357.

[5] S. M. Hezavehi, D. Weyns, P. Avgeriou, R. Calinescu, R. Mirandola, and D. Perez-Palacin, "Uncertainty in self-adaptive systems: A research community perspective," *ACM Trans. Auton. Adapt. Syst.*, vol. 15, no. 4, Dec. 2021. DOI: 10.1145/3487921.

[6] J. Cámara, J. Troya, A. Vallecillo, *et al.*, "The uncertainty interaction problem in self-adaptive systems," *Software and systems modeling*, vol. 21, no. 4, pp. 1277–1294, 2022.

[7] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, and M. Fisher, "Formal specification and verification of autonomous robotic systems: A survey," *ACM Comput. Surv.*, vol. 52, no. 5, Sep. 2019.

[8] P. Oreizy, N. Medvidovic, and R. Taylor, "Architecture-based runtime software evolution," in *Proceedings of the 20th International Conference on Software Engineering*, 1998, pp. 177–186. DOI: 10.1109/ICSE.1998.671114.

[9] D. Weyns, S. Malek, and J. Andersson, "Forms: A formal reference model for self-adaptation," in *Proceedings of the 7th International Conference on Autonomic Computing*, ser. ICAC '10, Washington, DC, USA: Association for Computing Machinery, 2010, pp. 205–214.

[10] H. Liang, J. S. Dong, J. Sun, and W. E. Wong, "Software monitoring through formal specification animation," *Innovations in systems and software engineering*, vol. 5, no. 4, pp. 231–241, 2009.

[11] A. Tarasyuk, I. Pereverzeva, E. Troubitsyna, T. Latvala, and L. Nummila, "Formal development and assessment of a reconfigurable on-board satellite system," in *Proceedings of the 31st International Conference on Computer Safety, Reliability, and Security*, ser. SAFECOMP'12, Magdeburg, Germany: Springer-Verlag, 2012, pp. 210–222.

[12] P. Izzo, H. Qu, and S. M. Veres, "A stochastically verifiable autonomous control architecture with reasoning,"

in *2016 IEEE 55th Conference on Decision and Control (CDC)*, 2016, pp. 4985–4991.

[13] M. Machin, F. Dufossé, J. Guiochet, D. Powell, M. Roy, and H. Waeselynck, "Model-checking and game theory for synthesis of safety rules," in *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*, 2015, pp. 36–43.

[14] M. Brambilla, A. Brutschy, M. Dorigo, and M. Birattari, "Property-driven design for robot swarms: A design method based on prescriptive modeling and model checking," *ACM Trans. Auton. Adapt. Syst.*, vol. 9, no. 4, Dec. 2014.

[15] S. Konur, C. Dixon, and M. Fisher, "Formal verification of probabilistic swarm behaviours," in *Swarm Intelligence*, M. Dorigo, M. Birattari, G. A. Di Caro, *et al.*, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 440–447.

[16] S. Mitsch, K. Ghorbal, and A. Platzer, "On provably safe obstacle avoidance for autonomous robotic ground vehicles," in *Robotics: Science and Systems IX, Technische Universität Berlin, Berlin, Germany, June 24 - June 28, 2013*, 2013.

[17] A. Platzer, "Differential dynamic logic for verifying parametric hybrid systems," in *Automated Reasoning with Analytic Tableaux and Related Methods*, N. Olivetti, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 216–232.

[18] M. Y. Vardi, "An automata-theoretic approach to linear temporal logic," in *Logics for Concurrency: Structure versus Automata*, F. Moller and G. Birtwistle, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 238–266.

[19] J. R. Celaya, A. A. Desrochers, and R. J. Graves, "Modeling and analysis of multi-agent systems using petri nets," in *2007 IEEE International Conference on Systems, Man and Cybernetics*, 2007, pp. 1439–1444.

[20] H. Costelha and P. Lima, "Modelling, analysis and execution of robotic tasks using petri nets," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 1449–1454.

[21] R. Halder, J. Proença, N. Macedo, and A. Santos, "Formal verification of ros-based robotic applications using timed-automata," in *2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering (FormaliSE)*, 2017, pp. 44–50.

[22] M. U. Iftikhar and D. Weyns, "A case study on formal verification of self-adaptive behaviors in a decentralized system," *Electronic Proceedings in Theoretical Computer Science*, vol. 91, pp. 45–62, Aug. 2012.

[23] N. Correll and A. Martinoli, "Modeling and designing self-organized aggregation in a swarm of miniature robots," *The International Journal of Robotics Research*, vol. 30, no. 5, pp. 615–626, 2011. DOI: 10.1177/0278364911403017.

[24] O. Michel, "Cyberbotics ltd. webots™: Professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004. DOI: 10.5772/5618.

[25] T. Gibson-Robinson, P. Armstrong, A. Boulgakov, and A. W. Roscoe, "Fdr3 — a modern refinement checker for csp," in *Tools and Algorithms for the Construction and Analysis of Systems*, E. Ábrahám and K. Havelund, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 187–201.

[26] T. Haidegger, M. Barreto, P. Gonçalves, *et al.*, "Applied ontologies and standards for service robots," *Robotics and autonomous systems*, vol. 61, no. 11, pp. 1215–1223, 2013.

[27] E. Prestes, J. L. Carbonera, S. Rama Fiorini, *et al.*, "Towards a core ontology for robotics and automation," *Rob. Auton. Syst.*, vol. 61, no. 11, pp. 1193–1204, Nov. 2013.

[28] N. Akhtar and M. M. S. Missen, "Contribution to the formal specification and verification of a multi-agent robotic system," *ArXiv*, vol. abs/1604.05577, 2015.

[29] E. M. Clarke, "Model checking," in *Foundations of Software Technology and Theoretical Computer Science*, S. Ramesh and G. Sivakumar, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 54–56, ISBN: 978-3-540-69659-9.

[30] M. Abdul Basit Ur Rahim, M. Ahsan Ur Raheem, M. K. Sohail, M. A. Farid, and M. R. Mufti, "Formal verification of reconfigurable systems," *Soft Comput.*, May 2023.

[31] M. A. B. ur Rahim and F. Arif, "Translating activity diagram from duration calculus for modeling of real-time systems and its formal verification using uppaal and divine, vol. 35 (1)," *Mehran University Research Journal of Engineering and Technolog, Berlin, Heidelberg*, pp. 139–154, 2016.

[32] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Uppaal — a tool suite for automatic verification of real-time systems," in *Hybrid Systems III*, R. Alur, T. A. Henzinger, and E. D. Sontag, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 232–243, ISBN: 978-3-540-68334-6.

[33] M. Abdul Basit Ur Rahim, Q. Duan, and E. Al-Shaer, "A formal analysis of moving target defense," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2020, pp. 1802–1807. DOI: 10.1109/COMPSAC48688.2020.00050.

[34] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi, "On the temporal analysis of fairness," in *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '80, Las Vegas, Nevada: Association for Computing Machinery, 1980, pp. 163–173.