Software Requirements Specification for 2D Robot Arms for Path Planning

Ziyang(Ryan) Fang

February 5, 2025

Contents

Revision History

Date	Version	Notes
Feb 02 2025	1.0	Notes

1 Reference Material

This section records information for easy reference.

1.1 Table of Units

Throughout this document SI (Système International d'Unités) is employed as the unit system. In addition to the basic units, several derived units are used as described below. For each unit, the symbol is given followed by a description of the unit and the SI name.

symbol	**unit**	**SI**
mm	length	millimetre
s	time	second
q	joint angle	radians
$\dot{q}$	joint velocity	rad/s
$\ddot{q}$	joint acceleration	rad/s^2

1.2 Table of Symbols

The following table provides a summary of the symbols used throughout this document, along with their respective units and descriptions. The choice of symbols aligns with conventions in robotics and motion planning literature to ensure consistency and clarity. Symbols are listed in alphabetical order for easy reference.

symbol	**unit**	**description**
L	mm	Link length of the robot arm
q	radians	Joint angle
$\dot{q}$	rad/s	Joint angular velocity
$\ddot{q}$	rad/s^2	Joint angular acceleration
x, y	mm	Cartesian coordinates of the end-effector
T	s	Total time for motion execution
t	s	Time step
$\mathbf{q}_{\text{init}}$	radians	Initial joint configuration
$\mathbf{q}_{\text{goal}}$	radians	Goal joint configuration
O_i	mm	Position of the i-th obstacle (circular)
r_i	mm	Radius of the i-th obstacle
θ_i	radians	Discretized joint angle for A* search

$\mathbf{p}_i$	mm	Position of the i-th joint

1.3 Abbreviations and Acronyms

symbol	description
A	Assumption
DD	Data Definition
GD	General Definition
GS	Goal Statement
IM	Instance Model
LC	Likely Change
PS	Physical System Description
R	Requirement
SRS	Software Requirements Specification
TM	Theoretical Model
IK	Inverse Kinematics
FK	Forward Kinematics
A*	A-star Pathfinding Algorithm
DOF	Degrees of Freedom
EE	End-Effector
TM	Theoretical Model
2D-RAPP	2D Robot Arm Path Planning

2 Introduction

Path planning for robotic manipulators is a fundamental problem in robotics, enabling robots to move efficiently and safely within constrained environments. Unlike mobile robots that navigate in free space, robotic arms must operate in a higher-dimensional joint space, where each degree of freedom contributes to the overall complexity of motion planning. Traditional inverse kinematics (IK) methods solve for joint angles that satisfy end-effector goals, but they often struggle with obstacle avoidance and computational efficiency in real-time applications.

This document provides an overview of the Software Requirements Specification (SRS) for a 2D robotic arm path planning system based on a graph-based inverse kinematics (IK) approach with A* search. The developed program will be referred to as 2D Robot Arm Path Planning (2D-RAPP) throughout this document. This section explains the purpose of this document, the scope of the requirements, the characteristics of the intended reader, and the organization of the document.

2.1 Purpose of Document

The primary purpose of this document is to record the requirements of 2D-RAPP. Goals, assumptions, theoretical models, definitions, and other derivation information are specified, allowing the reader to fully understand and verify the purpose and scientific basis of 2D-RAPP. With the exception of system constraints, this SRS remains abstract, describing what problem is being solved, but not how to solve it.

This document serves as a starting point for subsequent development phases, including writing the design specification and the software verification and validation plan. The design document will show how the requirements are to be realized, including decisions on numerical algorithms, data structures, and programming frameworks. The verification and validation plan will outline the steps required to increase confidence in the software's correctness, reliability, and performance.

Although this SRS follows a structured documentation approach, it does not impose any specific software development methodology. Regardless of whether a waterfall, agile, or iterative development process is followed, it remains useful to present the documentation in a way that maintains clarity and logical structure, as recommended by Parnas and Clements.

2.2 Scope of Requirements

The scope of the requirements includes path planning for a 2D robotic arm with N revolute joints, operating in a workspace with circular obstacles. The system is designed to compute collision-free paths in joint space, using distance-based inverse kinematics and search algorithms.

2.3 Characteristics of Intended Reader

Reviewers of this documentation should have a fundamental understanding of robotics kinematics at an undergraduate level, specifically covering forward kinematics, inverse kinematics, and path planning. Familiarity with graph-based search algorithms (such as A*) and numerical optimization techniques is also recommended.

The users of 2D-RAPP (e.g., robotics researchers, engineers) may have varying levels of expertise, as explained in Section 3.2. However, those involved in reviewing, modifying, or maintaining this SRS should have a strong background in computational kinematics and algorithmic path planning.

2.4 Organization of Document

This document follows the standard structure of an SRS for scientific computing software. It begins with the problem statement and goals, followed by theoretical foundations, definitions, and assumptions. The document then presents instance models and specific requirements.

The problem is introduced through kinematic models and path planning techniques, including forward and inverse kinematics and graph-based search methods. Assumptions

related to joint constraints and obstacle avoidance are also outlined.

Readers can approach this document either top-down, starting from goals to implementation, or bottom-up, beginning with instance models and tracing back to theoretical foundations. The instance model IM:PathPlanning defines the core algorithm for collision-free motion planning of a 2D robot arm.

3 General System Description

This section provides general information about the system. It identifies the interfaces between the system and its environment, describes the user characteristics and lists the system constraints.

3.1 System Context

Fig. 1 illustrates the system context. The circle represents the user, an external entity providing inputs to the system. The rectangle represents the software system, 2D-RAPP. Arrows indicate the data flow between the user and the system.

Figure 1: System Context

The interaction between the user and the system is described as follows:

User Responsibilities

- Provide the input data to the system, including:
  - Initial joint angles (q_{init}) to specify the starting configuration of the robotic arm.
  - Goal position ($x_{\text{goal}}, y_{\text{goal}}$) to indicate the desired end-effector target.
  - Obstacle data, including radii (r_i) and positions ($O_i = (x_i, y_i)$).
- Ensure the input data is accurate and in the required units.

2D-RAPP Responsibilities

- Detect input errors, such as invalid data types or inconsistent units.
- Verify that the inputs satisfy physical and software constraints.
- Compute a collision-free trajectory for the robotic arm, including:
  - Joint angles (q, $\dot{q}$, $\ddot{q}$) for each motion stage.
  - Cartesian path (x, y) of the end-effector.
- Provide a visual representation of the robotic arm's movement and obstacle avoidance.
- Detect unreachable goal positions:
  - If the target position is not reachable due to kinematic limitations or obstacles, return an empty result.
  - Notify the user that the goal position cannot be reached.

This system context defines the roles and interactions between the user and the 2D-RAPP system, ensuring clarity in the data flow and system responsibilities.

3.2 User Characteristics

The intended users of 2D-RAPP are individuals with a background in robotics, computer science, and mechanical engineering. They are expected to have a fundamental understanding of kinematics, particularly inverse kinematics, and be familiar with basic path planning algorithms.

3.3 System Constraints

There are no system constraints.

4 Specific System Description

This section first presents the problem description, which gives a high-level view of the problem to be solved. This is followed by the solution characteristics specification, which presents the assumptions, theories, definitions and finally the instance models. The system focuses on generating a collision-free trajectory for a 2D robotic arm in an environment with static obstacles. The robotic arm is modeled as a serial manipulator with revolute joints, and its configuration is defined by a set of joint angles.

4.1 Problem Description

2D-RAPP is intended to solve the path planning for 2D robotic arms.

4.1.1 Terminology and Definitions

This subsection provides a list of terms that are used in the subsequent sections and their meaning, with the purpose of reducing ambiguity and making it easier to correctly understand the requirements:

- **Joint space:** The space defined by the possible values of the robot arm's joint angles.
- **Workspace:** The Cartesian space where the end-effector of the robotic arm can reach.
- **Obstacle:** A static circular object in the workspace that the robotic arm must avoid.
- **Trajectory:** A sequence of joint configurations representing a smooth motion of the robotic arm from the start to the goal.
- **Configuration:** A specific set of joint angles that fully describes the position of the robotic arm.
- **Collision detection:** The process of determining whether the robotic arm intersects with an obstacle.
- **Path planning:** The process of computing a collision-free sequence of configurations to move the robotic arm from the start to the goal.
- **Inverse kinematics:** The process of determining joint angles given a desired end-effector position.
- **Forward kinematics:** The computation of the end-effector position given joint angles.

4.1.2 Physical System Description

The physical system of 2D-RAPP, as shown in Figure 2, includes the following elements:

PS1: A robotic arm with N revolute joints, defining a N-link planar mechanism.

PS2: The environment contains static circular obstacles that must be avoided by the arm.

PS3: A discretized joint space where joint angles are sampled uniformly to form a toroidal search space.

PS4: A start configuration that represents the initial joint angles of the robotic arm.

PS5: A goal configuration that represents the desired joint angles to reach the target position.

PS6: The A* algorithm used to compute the shortest collision-free path in joint space.

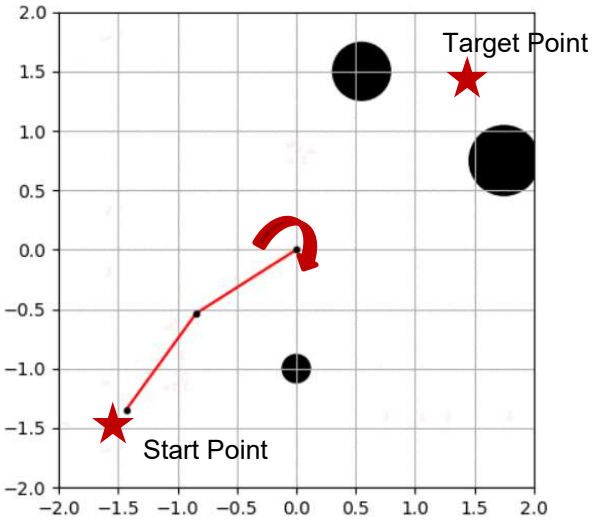


Figure 2: Robotic arm finding a path to target point

4.1.3 Goal Statements

Given the initial joint angles, the goal joint angles, and the obstacle locations, the goal statements are:

GS1: Compute a collision-free trajectory in joint space from the start configuration to the goal configuration.

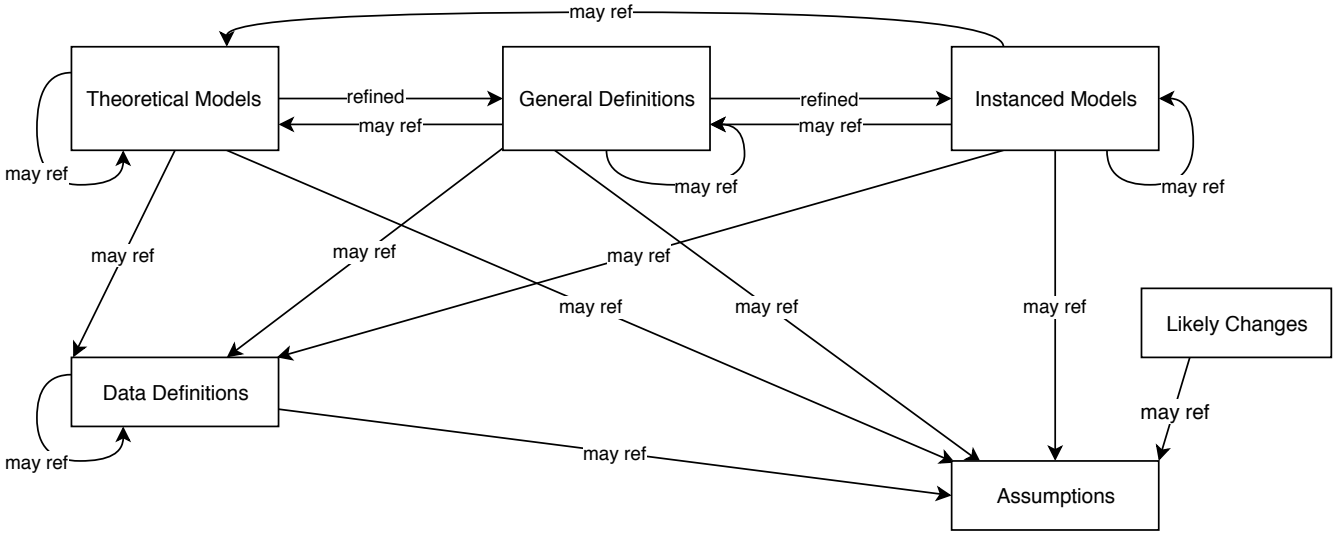
GS2: Minimize the total path length while ensuring obstacle avoidance.

GS3: Generate a smooth and continuous motion trajectory for the robotic arm.

GS4: Validate the safety of the planned path by verifying obstacle-free movement at each configuration.

4.2 Solution Characteristics Specification

The instance models that govern 2D-RAPP are presented in the Instance Model Section. The necessary information to understand the meaning of the instance models and their derivation is also included, ensuring that the instance models can be verified. The solution characteristics describe the assumptions, theoretical foundations, and mathematical formulations that define the problem space and provide a basis for trajectory planning and obstacle avoidance in robotic arm motion.



The instance models that govern 2D-RAPP are presented in Subsection 4.2.6. The information to understand the meaning of the instance models and their derivation is also presented, so that the instance models can be verified.

4.2.1 Assumptions

This section simplifies the original problem and helps in developing the theoretical model by filling in the missing information for the physical system. The numbers given in the square brackets refer to the theoretical model [TM], general definition [GD], data definition [DD], instance model [IM], or likely change [LC], in which the respective assumption is used.

A1: The motion of the robotic arm is modeled purely kinematically, without considering the effects of dynamics such as forces, torques, or friction. (RefBy: IM:PathPlanning)

A2: The lengths of all links in the robotic arm are constant and do not change during motion. (RefBy: GD:ForwardKinematics)

A3: The joint angles of the robotic arm vary smoothly and continuously without sudden jumps. (RefBy: IM:TrajectoryGeneration)

A4: The robotic arm links are considered rigid, meaning there is no mechanical deflection or deformation. (RefBy: GD:InverseKinematics)

A5: Obstacles in the environment are approximated as circles with predefined radii. (RefBy: IM:CollisionDetection)

A6: The planned trajectory is discretized into a series of intermediate waypoints in joint space. (RefBy: IM:TrajectoryGeneration)

A7: The initial joint configuration and the goal joint configuration are known before path planning begins. (RefBy: IM:PathPlanning)

4.2.2 Theoretical Models

This section focuses on the general equations and laws that 2D-RAPP is based on. The theoretical models serve as fundamental principles governing the kinematic motion and path planning of the robotic arm.

RefName: **TM:FWDKinematics**

Label: Forward Kinematics

Equation: $\mathbf{x} = f(\mathbf{q}) = \begin{bmatrix} L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2) \\ L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2) \end{bmatrix}$

Description:

- $\mathbf{q}$: Joint angles, where θ_1 is the angle of the first joint and θ_2 is the relative angle of the second joint.
- L_1: Length of the first link (mm).
- L_2: Length of the second link (mm).

Notes: This forward kinematics model assumes:

- Rigid links: The links are assumed to be perfectly rigid and do not deform under load (A2).
- Fixed base: The base of the arm is considered stationary, located at the origin $(0, 0)$.
- Ideal joints: The joints rotate without friction or other physical limitations.
- Planar motion: The arm operates in a 2D plane, and there is no out-of-plane movement.

These assumptions ensure simplicity and accuracy for theoretical modeling. For real-world applications, additional factors such as joint compliance, link flexibility, and external forces may need to be considered.

Source: https://atsushisakai.github.io/PythonRobotics/modules/7_arm_navigation/planar_two_link_ik.html

Ref. By: GD??

Preconditions for TM:FWDKinematics: None

Derivation for TM:FWDKinematics: Not Applicable

4.2.3 General Definitions

This section collects the laws and equations that will be used in building the instance models.

Number	GD1
Label	**Forward Kinematics of a 2D Robotic Arm**
SI Units	mm
Equation	$$x = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2)$$ $$y = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2)$$
Description	The forward kinematics equations describe the position of the end-effector in Cartesian coordinates given the joint angles and link lengths. x, y are the Cartesian coordinates of the end-effector (mm). L_1, L_2 are the lengths of the arm's links (mm). θ_1, θ_2 are the joint angles (radians). Assumptions: • The links are rigid and have fixed lengths (A2). • The joints rotate freely within the plane. • The base of the robotic arm is fixed.
Source	–

Detailed derivation of forward kinematics equations

The forward kinematics of a 2D robotic arm is derived by considering the arm as a series of rigid links connected by rotational joints. The position of the end-effector is computed as follows:

1. The first joint rotates by an angle θ_1, positioning the second joint at:

$$x_1 = L_1 \cos(\theta_1), \quad y_1 = L_1 \sin(\theta_1).$$

2. The second joint rotates by an angle θ_2, contributing to the final position:

$$x = x_1 + L_2 \cos(\theta_1 + \theta_2), \quad y = y_1 + L_2 \sin(\theta_1 + \theta_2).$$

3. Substituting x_1 and y_1 results in the final equations:

$$x = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2),$$

$$y = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2).$$

These equations provide a direct mapping from joint space to Cartesian space, which is essential for path planning and trajectory control in robotic applications.

4.2.4 Data Definitions

This section defines and collects the essential data needed to build the instance models. The variables defined here will be used throughout the document.

Number	DD1
Label	**Forward Kinematics Data Definition**
Symbol	$\mathbf{x} = (x, y)$
SI Units	mm
Equation	$x = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2)$ $y = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2)$
Description	Forward kinematics describes how the end-effector position in Cartesian coordinates (x, y) is computed as a function of the joint angles θ_1, θ_2 and link lengths L_1, L_2. • x, y: Cartesian coordinates of the end-effector (mm). • L_1, L_2: Lengths of the robotic arm's links (mm). • θ_1, θ_2: Joint angles (radians). This assumes: • The robotic arm operates in a 2D plane. • The links are rigid with fixed lengths (A2). • The base of the robotic arm is fixed at $(0, 0)$.
Source	Robotics textbook or relevant reference
Ref. By	GD??, GD??

4.2.5 Data Types

This section defines the data types used in the robotic arm path planning (2D-RAPP) system. These types are used to specify inputs, intermediate values, and outputs in the models.

Type Name	Description
`Angle`	Represents joint angles of the robotic arm in radians.
`Length`	Represents physical lengths (e.g., arm links, obstacle radii) in millsmeters.
`Position`	A 2D Cartesian coordinate represented as (x, y) in millsmeters.
`Velocity`	Represents joint angular velocity in radians per second.
`Acceleration`	Represents joint angular acceleration in radians per second squared.
`Trajectory`	A sequence of joint angles describing the motion plan.
`Obstacle`	A structure containing an obstacle's center position (x, y) and radius.
`Boolean`	A binary value (true/false) indicating feasibility of a path.

4.2.6 Instance Models

This section transforms the problem defined in Section 4.1 into one expressed in mathematical terms. It uses concrete symbols defined in Section 4.2.4 to replace the abstract symbols in the models identified in Sections 4.2.2 and 4.2.3.

The goal statements 4.1.3 are addressed by the following instance models:

RefName: **IM:TrajectoryGeneration**

Label: Trajectory Generation

Equation: $\mathbf{q}(t) = \text{PathPlanner}(\mathbf{q}_{\text{init}}, \mathbf{q}_{\text{goal}}, O, r)$

Description: The trajectory generation model computes a smooth, collision-free sequence of joint angles $\mathbf{q}(t)$ (radians) to move the robotic arm from the initial configuration $\mathbf{q}_{\text{init}}$ to the goal configuration $\mathbf{q}_{\text{goal}}$. The function PathPlanner considers obstacle locations O_i and radii r_i to avoid collisions.

The planned trajectory satisfies the following conditions:

- The path is kinematically feasible, considering joint limits and maximum velocities.
- No path segment intersects any obstacle.
- If no valid path exists, the system returns an empty trajectory and an infeasibility warning.

Notes: This model assumes a 2D planar manipulator with revolute joints and no external disturbances (A**??**).

Source: https://www.robotics.org/path_planning

Ref. By: R3, R5

Preconditions for IM:TrajectoryGeneration: The start and goal configurations must be valid joint states within the robot's joint limits.

Derivation for IM:TrajectoryGeneration: Not Applicable

RefName: **IM:ObstacleAvoidance**

Label: Obstacle Avoidance

Equation: $\min_i \|\mathbf{x}(t) - O_i\| \geq r_i + \epsilon, \quad \forall t$

Description: This model ensures that the robotic arm does not collide with obstacles during motion. The distance between the end-effector position $\mathbf{x}(t)$ and each obstacle center O_i must always be greater than the obstacle radius r_i plus a safety margin ϵ.

This condition is checked at every time step of the planned trajectory. If violated, a new trajectory is generated, or an infeasibility warning is returned.

Notes: This model assumes that obstacle positions remain static during execution (A**??**).

Source: https://www.robotics.org/collision_avoidance

Ref. By: R3, R4

Preconditions for IM:ObstacleAvoidance: The obstacle positions and radii must be known before trajectory planning.

Derivation for IM:ObstacleAvoidance: Not Applicable

Derivation of Trajectory Generation (IM:TrajectoryGeneration)

The trajectory generation model is derived from forward kinematics and inverse kinematics principles.

1. **Forward Kinematics:** The position of the end-effector $\mathbf{x}(t)$ is determined using:

$$x = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2)$$

$$y = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2)$$

This establishes the mapping from joint space $\mathbf{q}(t)$ to Cartesian space.

2. **Inverse Kinematics:** Given a desired goal position $\mathbf{x}_{\text{goal}} = (x_g, y_g)$, we solve for the required joint angles:

$$\theta_2 = \cos^{-1}\left(\frac{x_g^2 + y_g^2 - L_1^2 - L_2^2}{2L_1L_2}\right)$$

$$\theta_1 = \tan^{-1}\left(\frac{y_g}{x_g}\right) - \tan^{-1}\left(\frac{L_2 \sin\theta_2}{L_1 + L_2\cos\theta_2}\right)$$

   This establishes the mapping from Cartesian space to joint space.

3. **Path Planning:** A trajectory $\mathbf{q}(t)$ is generated using an interpolation method such as:

   - Linear interpolation (for simple movements).
   - Polynomial interpolation (e.g., cubic splines) for smooth acceleration profiles.
   - Graph-based search algorithms (e.g., A* on a discretized joint space).

4. **Collision Avoidance:** Each planned trajectory segment is verified against obstacles using:

$$\min_i \|\mathbf{x}(t) - O_i\| \geq r_i + \epsilon, \quad \forall t$$

   If this constraint is violated, a new path is computed, or an infeasibility warning is returned.

The derivation combines these principles to ensure a feasible, collision-free trajectory for the robotic arm.

Derivation of Obstacle Avoidance (IM:ObstacleAvoidance)

Obstacle avoidance is derived from the Euclidean distance constraint:

1. **Obstacle Representation:** Obstacles are modeled as circles with centers $O_i = (O_{i_x}, O_{i_y})$ and radii r_i.

2. **Collision Condition:** A collision occurs if:

$$\|\mathbf{x}(t) - O_i\| < r_i$$

   where $\mathbf{x}(t)$ is the end-effector position.

3. **Safe Clearance Constraint:** To avoid collisions, the trajectory must satisfy:

$$\|\mathbf{x}(t) - O_i\| \geq r_i + \epsilon, \quad \forall t$$

   where ϵ is a safety margin.

4. **Validation:** Each trajectory segment is checked at discrete time steps. If a collision is detected, the path planner reattempts a new trajectory.

This derivation ensures that the robotic arm follows a valid path while maintaining a safe distance from obstacles.

4.2.7 Input Data Constraints

Table 2 shows the data constraints on the input variables. The column for physical constraints gives the physical limitations on the range of values that can be taken by the variable. The column for software constraints restricts the range of inputs to reasonable values. These software constraints will be helpful in the design stage for selecting suitable algorithms. The constraints are conservative to give users flexibility in experimenting with unusual situations. The column of typical values provides a reference for common scenarios. The uncertainty column estimates the confidence with which the physical quantities can be measured. This information is useful for uncertainty quantification exercises.

The specification parameters in Table 2 are listed in Table 4.

Table 2: Input Variables for 2D-RAPP

Var	Physical Constraints	Software Constraints	Typical Value	Uncertainty
L_1, L_2	$L_i > 0$	$L_{\min} \leq L_i \leq L_{\max}$	1.0 m	5%
θ_1, θ_2	$-\pi \leq \theta_i \leq \pi$	$-\pi \leq \theta_i \leq \pi$	0 rad	1%
$x_{\text{goal}}, y_{\text{goal}}$	Defined workspace bounds	Within reachable workspace	(1.5, 1.5) m	10%
r_{obs}	$r_{\text{obs}} > 0$	$r_{\text{obs}} \leq r_{\max}$	0.3 m	5%
$(x_{\text{obs}}, y_{\text{obs}})$	Within workspace bounds	Within workspace bounds	(0.5, 0.5) m	10%

(*) θ_1, θ_2 are the joint angles of the robotic arm.

(*) L_1, L_2 are the lengths of the arm segments.

(*) $x_{\text{goal}}, y_{\text{goal}}$ define the Cartesian coordinates of the target.

(*) $(x_{\text{obs}}, y_{\text{obs}})$ and r_{obs} define the position and size of obstacles.

Table 4: Specification Parameter Values for 2D-RAPP

Var	Value
$L_{\min}$	0.1 m
$L_{\max}$	2.0 m
$r_{\max}$	0.5 m

4.2.8 Properties of a Correct Solution

A correct solution must exhibit the following properties to ensure robustness and feasibility in robotic arm path planning:

- The generated trajectory must be **collision-free**, ensuring that the robotic arm does not intersect any obstacles.
- The computed joint angles (θ_1, θ_2) must respect the physical limits of the robotic arm.
- The path must be **kinematically feasible**, meaning it can be executed smoothly by the robotic system.
- If the target position is unreachable due to obstacles or arm constraints, the system must return an empty trajectory and notify the user.

Additionally, output values must adhere to known physical constraints, summarized in Table 6.

Table 6: Output Variables and Physical Constraints

Var	Physical Constraints
θ_1, θ_2	$-\pi \leq \theta_1, \theta_2 \leq \pi$ (Joint limits)
(x, y)	Within reachable workspace of arm
$\mathbf{q}_{\text{path}}$	Must be collision-free with obstacles
$\mathbf{x}_{\text{goal}}$	If unreachable, return empty trajectory

These constraints ensure that the computed motion plan is valid, achievable, and safe for execution by the robotic arm.

5 Requirements

This section provides the functional requirements, the business tasks that the software is expected to complete, and the nonfunctional requirements, the qualities that the software is expected to exhibit.

5.1 Functional Requirements

R1: The system shall accept the following inputs from the user:

- Initial joint angles $q_{\text{init}} = (\theta_1, \theta_2)$ in radians.

- Goal position $(x_{\text{goal}}, y_{\text{goal}})$ in Cartesian coordinates (mm).
- Obstacle positions $O_i = (x_i, y_i)$ and radii r_i (mm).
- Link lengths L_1, L_2 of the robotic arm (mm).

R2: The system shall echo all input values to ensure correctness before computation.

R3: The system shall perform the following calculations:

- Compute the forward kinematics to determine the end-effector position.
- Determine the collision-free path from q_{init} to the goal using A* search.
- Generate the sequence of intermediate joint configurations $q(t)$ for smooth motion.

R4: The system shall verify that:

- The computed path avoids all obstacles.
- The final joint configuration corresponds to the given goal position.
- The output trajectory is continuous and feasible for the robotic arm.

R5: The system shall output the following:

- The computed path in joint space, represented as a sequence of joint angles $q(t)$.
- The Cartesian positions of the end-effector throughout the motion.
- A visualization of the planned motion with the robotic arm and obstacles.

5.2 Nonfunctional Requirements

This section defines the nonfunctional requirements that specify the qualities and characteristics the software should exhibit.

NFR1: **Accuracy**
The system shall compute the robotic arm trajectory with an accuracy sufficient for motion planning in a dynamic environment. The precision of the joint angles shall be within 0.01 radians, and the calculated Cartesian positions of the end-effector shall have an error margin within 0.1 cm. The accuracy shall be validated through simulation and empirical testing.

NFR2: **Maintainability**
The effort required to make likely modifications to the software (such as adjusting the planning algorithm or adding new constraints) shall be at most 10% of the original development time, assuming the same development resources are available. Code modularization and documentation will facilitate future modifications.

NFR3: **Verifiability**
The correctness of the system shall be tested with a complete verification and validation (V&V) plan. The outputs shall be compared against theoretical models and experimental data.

NFR4: **Understandability**
The software shall follow a modular architecture with clear documentation, including a module guide and interface specification. Each function and class shall have inline comments explaining its purpose and usage.a

NFR5: **Reusability**
The system shall be designed with reusability in mind, allowing for easy adaptation to different robotic configurations. The kinematic and planning modules shall be abstracted so they can be reused in other robotic applications.

5.3 Rationale

This section provides justifications for the decisions made in the documentation, including scope choices, modeling approaches, assumptions, and typical values.

- **Scope Decisions:** The scope is limited to 2D kinematic motion planning for a two-link robotic arm. This choice simplifies computations by avoiding the complexities of dynamic control and multi-degree-of-freedom motion. The system focuses on trajectory planning and collision avoidance, which are critical for robotic applications in structured environments.

- **Modeling Decisions:** The system models the robotic arm using forward kinematics and inverse kinematics equations. The choice of the A* algorithm for trajectory planning ensures optimal pathfinding in discrete joint space. A toroidal occupancy grid is used to handle the periodic nature of joint angles while maintaining computational efficiency.

- **Assumptions:** Several assumptions were made to reduce computational complexity and improve efficiency:

  - The robotic arm consists of two rigid links with fixed lengths (A2).
  - The motion occurs strictly in a 2D plane, ignoring dynamic effects such as inertia and external forces.
  - The joint angles are constrained within $[-\pi, \pi]$ to ensure practical movement constraints.

- **Typical Values:** The choice of typical values for input parameters is based on real-world robotic systems:

  - Link lengths (mm): $L_1 = 50$, $L_2 = 40$

- Joint angle range (rad): $[-\pi, \pi]$
- Obstacle radii (mm): $r_i \in [5, 15]$

These values ensure a balance between computational efficiency and practical usability.

6 Likely Changes

LC1: The resolution of the discrete grid used for A* search may need to be adjusted for better accuracy or computational efficiency. A finer grid provides more precise motion planning but increases computation time. (A6)

LC2: Currently, obstacles are modeled as circles for simplicity. Future iterations may need to support arbitrary polygonal obstacles to better reflect real-world scenarios. (A5)

LC3: The path-planning algorithm may need to be changed. Alternatives such as RRT (Rapidly-exploring Random Tree) or D* Lite might be required for more complex environments or real-time applications.

LC4: If the system is extended to 3D, the forward kinematics equations will need to be updated to account for additional degrees of freedom.

LC5: The current joint angle limits assume a symmetric range of motion. Some robotic arms have mechanical constraints that may require asymmetric limits, which would need to be accounted for in the calculations.

7 Unlikely Changes

LC6: The link lengths of the robotic arm are assumed to be fixed. Future versions are not expected to support variable-length arms or telescopic links. (A2)

LC7: The system is based purely on kinematics and does not consider dynamics (forces, torques, or inertia). It is unlikely that dynamics will be incorporated in this version. (A1)

LC8: The collision detection method assumes static obstacles. It is unlikely that dynamic obstacles (moving objects) will be introduced in this implementation. (A5)

8 Traceability Matrices and Graphs

The purpose of the traceability matrices is to provide easy references on what has to be additionally modified if a certain component is changed. Every time a component is changed, the items in the column of that component that are marked with an "X" may have to be modified as well. Table 8 shows the dependencies of theoretical models, general definitions, data definitions, and instance models with each other. Table 9 shows the dependencies of instance models, requirements, and data constraints on each other. Table **??** shows the dependencies of theoretical models, general definitions, data definitions, instance models, and likely changes on the assumptions.

	TM??	TM??	TM??	GD??	GD??	DD??	DD??	DD??	DD??	IM??	IM??	IM?
TM??												
TM??			X									
TM??												
GD??												
GD??	X											
DD??				X								
DD??				X								
DD??												
DD??								X				
IM??					X	X	X				X	
IM??					X		X		X	X		
IM??		X										
IM??		X	X				X	X	X		X	

Table 8: Traceability Matrix Showing the Connections Between Items of Different Sections

The purpose of the traceability graphs is also to provide easy references on what has to be additionally modified if a certain component is changed. The arrows in the graphs represent dependencies. The component at the tail of an arrow is depended on by the component at the head of that arrow. Therefore, if a component is changed, the components that it points to should also be changed. Figure **??** shows the dependencies of theoretical models, general definitions, data definitions, instance models, likely changes, and assumptions on each other. Figure **??** shows the dependencies of instance models, requirements, and data constraints on each other.

	IM??	IM??	IM??	IM??	4.2.7	R??	R??
IM??		X				X	X
IM??	X			X		X	X
IM??						X	X
IM??		X				X	X
R??							
R??						X	
R??					X		
R2	X	X				X	X
R??	X						
R??		X					
R??			X				
R??				X			
R4			X	X			
R??		X					
R??		X					

Table 9: Traceability Matrix Showing the Connections Between Requirements and Instance Models

	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??
IM??				X	X	X													

9 Development Plan

The development of 2D-RAPP will be structured into multiple phases, ensuring a modular and iterative approach. The implementation will prioritize core functionalities essential for the robotic arm's motion planning, followed by enhancements for efficiency and usability.

9.1 Phase 1: Core Functionalities

R1: Implement user input handling for joint angles, goal position, and obstacles.

R3: Develop forward kinematics calculations to determine the end-effector position from joint angles.

R5: Display the calculated position of the end-effector.

R4: Verify the correctness of kinematic calculations.

9.2 Phase 2: Path Planning

R3: Implement inverse kinematics to compute joint angles for a given target position.

R3: Integrate A* search algorithm for path planning in joint space.

R4: Ensure path validity by checking for obstacle collisions.

9.3 Phase 3: Optimization and Usability Enhancements

NFR1: Improve accuracy of kinematic computations.

NFR2: Ensure modularity in code for easy extension.

10 Values of Auxiliary Constants

The following table defines symbolic constants introduced in the document for maintainability and ease of modification.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Symbol	Description	Value
L_1	Length of first arm link (mm)	50
L_2	Length of second arm link (mm)	40
$\theta_{\min}$	Minimum joint angle (°)	-180
$\theta_{\max}$	Maximum joint angle (°)	180
Δq	Discretization step for A* search (radians)	0.1
FRACTION	Fraction of original development time for maintainability	0.1

Table 10: Values of Auxiliary Constants

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. How many of your requirements were inspired by speaking to your client(s) or their proxies (e.g. your peers, stakeholders, potential users)?

4. Which of the courses you have taken, or are currently taking, will help your team to be successful with your capstone project.

5. What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project? Examples of possible knowledge to acquire include domain specific knowledge from the domain of your application, or software engineering knowledge, mechatronics knowledge or computer science knowledge. Skills may be related to technology, or writing, or presentation, or team management, etc. You should look to identify at least one item for each team member.

6. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?