Software Requirements Specification for 2D Robot Arms for Path Planning

Ziyang(Ryan) Fang

April 19, 2025

Contents

Revision History

Date	Version	Notes
Feb 02 2025	1.0	Notes
April 5 2025	2.0	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	**Description**	**SI Unit**
m	length	metre (m)
s	time	second (s)
rad	angle	radian (rad)

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	m	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	m	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	m	Position of the i-th obstacle (circular)
r_i	m	Radius of the i-th obstacle
θ_i	radians	Discretized joint angle for A* search
$\mathbf{p}_i$	m	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
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.

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 foun-

dations. 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.

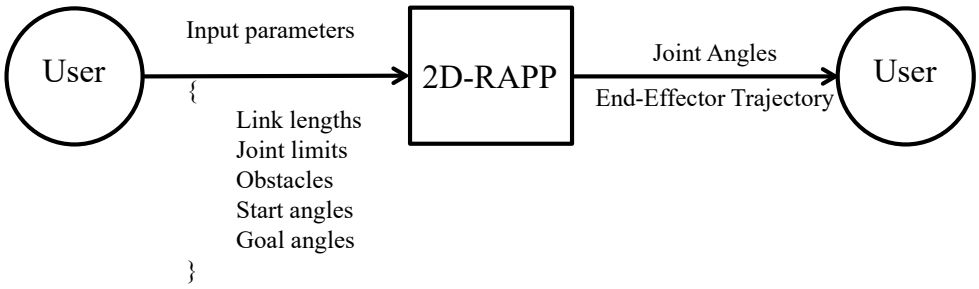


Figure 1: System Context

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

User Responsibilities

- Provide the input data to the system, including:
  - **Link lengths** ($L = [l_1, l_2, \ldots, l_n]$): a list of positive real numbers specifying the length of each robotic arm segment.
  - **Joint limits** ($J = [(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)]$): the allowable angle range for each joint.
  - **Obstacle data**: a list of obstacles defined by their center positions and radii: $O_i = (x_i, y_i, r_i)$.
  - **Start configuration** ($q_{\text{start}} \in \mathbb{R}^n$): the initial joint angles of the robot in radians.
  - **Goal configuration** ($q_{\text{goal}} \in \mathbb{R}^n$): the target joint angles to be reached.
- Ensure the input data is accurate and in the required unit

2D-RAPP Responsibilities

- Detect input errors, such as invalid data types, joint angle configurations outside of specified limits, or inconsistent angle units (e.g., degrees vs radians). Proper error messages are displayed via the GUI to guide user corrections.
- 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 at least undergraduate-level understanding of forward and inverse kinematics, as well as exposure to basic path planning algorithms such as A*, RRT, or Dijkstra.

3.3 System Constraints

The system uses A* search on a toroidal configuration space for path planning. This decision constrains the path planner to discrete grids and admissible heuristics, and may limit real-time scalability for high-dimensional arms. Other planning algorithms such as RRT or PRM could be integrated in future versions.

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.

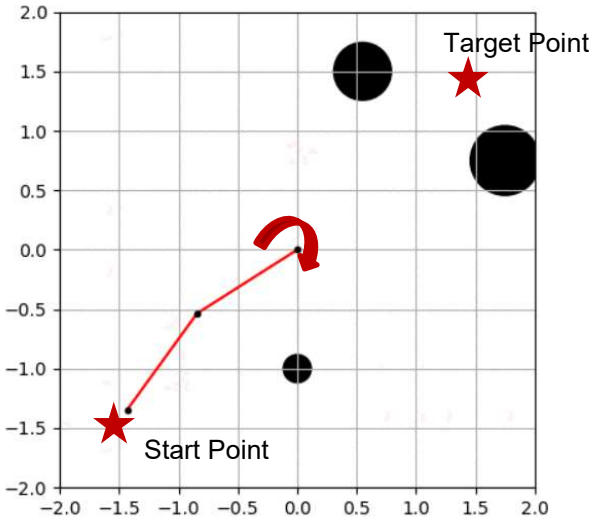


Figure 2: Robotic arm finding a path to target point

4.1.3 Goal Statements

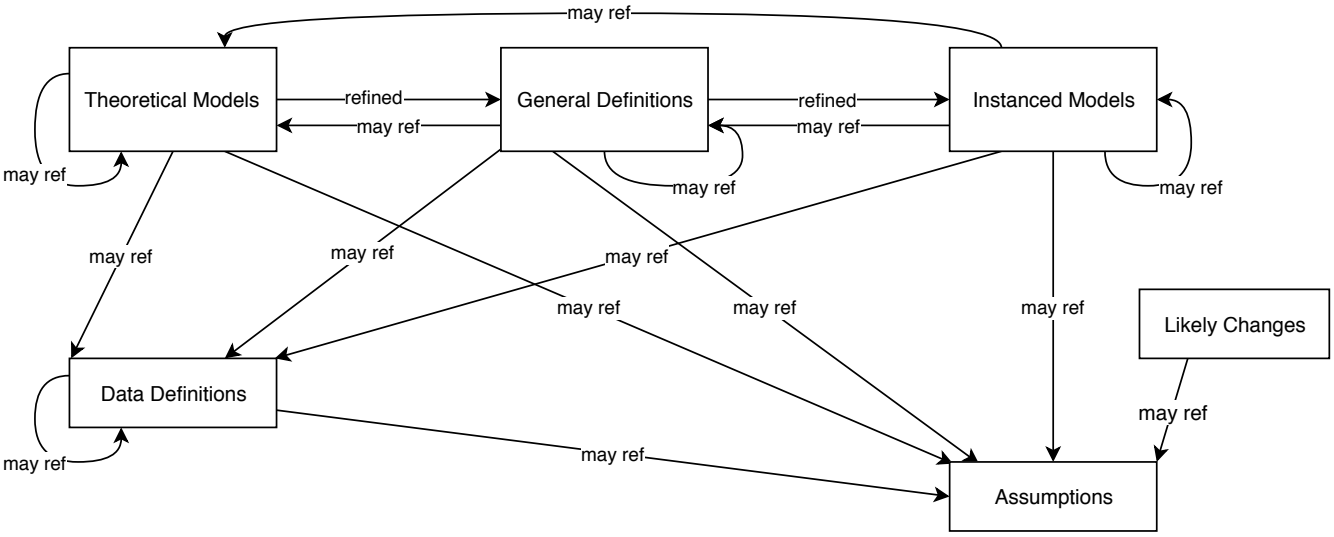
Given the initial joint angles, the goal joint angles, and the obstacle locations, the goal statements are:

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

GS2: Minimize the number of joint-space steps between start and goal configurations, each being a collision-free discrete 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:

$$\mathbf{x} = f(\mathbf{q}) = \begin{bmatrix} \sum_{i=1}^{n} L_i \cos\left(\sum_{j=1}^{i} \theta_j\right) \\ \sum_{i=1}^{n} L_i \sin\left(\sum_{j=1}^{i} \theta_j\right) \end{bmatrix}$$

Description: This equation describes the forward kinematics function f, mapping the joint angles $\mathbf{q}$ to the Cartesian position $\mathbf{x}$ of the robotic arm's end-effector for an n-link planar manipulator.

- n: The number of links in the robotic arm.
- $\mathbf{q} = [\theta_1, \theta_2, \ldots, \theta_n]^T$: The vector of joint angles, where θ_i is the relative angle of joint i.
- $\mathbf{x} = [x, y]^T$: Cartesian position of the end-effector (in m), computed relative to the arm's base at the origin.
- f: Forward kinematics mapping function, converting joint angles ($\mathbf{q}$) into the end-effector position ($\mathbf{x}$).
- L_i: Length of link i (m), connecting joint i to joint $(i+1)$.

This general formulation supports robotic arms with an arbitrary number of joints, enabling flexibility for various planar manipulator configurations.

Notes: This forward kinematics model assumes:

- Rigid links: All links are perfectly rigid and do not deform under load (A2).
- Fixed base: The robotic arm base is stationary, positioned at the origin $(0, 0)$.
- Planar motion: All arm movements are restricted to a 2D plane without out-of-plane motion.

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

Ref. By: GD1

Preconditions for TM:FWDKinematics: None

Derivation for TM:FWDKinematics: Not Applicable

RefName: **TM:SegmentCircleCollision**

Label: Collision Detection Between Line Segment and Circular Obstacle

Equation: Let the endpoints of a link segment be $\mathbf{a} = [x_1, y_1]$ and $\mathbf{b} = [x_2, y_2]$, and the circular obstacle have center $\mathbf{c} = [c_x, c_y]$ with radius r. Define the closest point $\mathbf{p}$ on the segment to the circle center as:

$$t = \frac{(\mathbf{c} - \mathbf{a}) \cdot (\mathbf{b} - \mathbf{a})}{\|\mathbf{b} - \mathbf{a}\|^2}, \quad t' = \text{clip}(t, 0, 1), \quad \mathbf{p} = \mathbf{a} + t'(\mathbf{b} - \mathbf{a})$$

Then collision occurs if:

$$\|\mathbf{p} - \mathbf{c}\| \leq r$$

Description: This theoretical model determines whether a robotic arm link, modeled as a line segment, intersects (collides) with a circular obstacle. It uses geometric projection to find the shortest distance from the circle center to the segment, checking if it is less than or equal to the obstacle's radius.

- $\mathbf{a}, \mathbf{b}$: Endpoints of the line segment representing the robotic arm link (in Cartesian coordinates, m).
- $\mathbf{c}$: Center of the circular obstacle (Cartesian coordinates, m).
- r: Radius of the circular obstacle (m).
- t': Clipped scalar projection factor ensuring the closest point remains within the segment bounds.
- $\mathbf{p}$: Closest point on the segment to the circle center.
- t': Clipped scalar projection factor ensuring the closest point remains within the segment bounds. Specifically, t' is constrained to $[0, 1]$ such that

$$t' = \begin{cases} 0 & \text{if } t < 0 \\ t & \text{if } 0 \leq t \leq 1 \\ 1 & \text{if } t > 1 \end{cases}$$

This method is computationally efficient and accurate for planar collision detection between line segments and circular obstacles.

Notes: This model assumes:

- The arm links are perfectly rigid and represented as straight line segments (A2).
- Obstacles are modeled as perfect circles.
- The robotic arm operates exclusively in a 2D plane without deformation.

Source: Adapted from collision detection concepts in computational geometry: https://www.geometrictools.com/Documentation/DistancePointLine.pdf

Ref. By: GD1

Preconditions for TM:SegmentCircleCollision: None

Derivation for TM:SegmentCircleCollision: Not Applicable

RefName: **TM:AStarJointSpace**

Label: A* Search Algorithm in Discretized Toroidal Joint Space

Equation: The A* search algorithm finds the optimal path between start node $\mathbf{s}$ and goal node $\mathbf{g}$ in an n-dimensional discrete joint space of resolution M per dimension. The evaluation function is defined as:

$$f(\mathbf{n}) = g(\mathbf{n}) + h(\mathbf{n})$$

where:

$$g(\mathbf{n}) = \text{cost from start to node } \mathbf{n}, \quad h(\mathbf{n}) = \sum_{i=1}^{n} \min\left(|n_i - g_i|, M - |n_i - g_i|\right)$$

Description: This theoretical model describes the A* search algorithm adapted to a toroidal (wrap-around) discretized joint configuration space for robotic arm path planning. The heuristic $h(n)$ used is the toroidal Manhattan distance, providing an admissible heuristic suitable for discrete, periodic search spaces.

- $\mathbf{n} = [n_1, n_2, \ldots, n_n]$: Current node indices in the discretized joint-space grid.
- $\mathbf{s}$: Start configuration node indices.
- $\mathbf{g}$: Goal configuration node indices.
- M: Number of discrete intervals per joint dimension.
- $g(\mathbf{n})$: Actual cost from the start to the node $\mathbf{n}$ (each step has uniform cost = 1).
- $h(\mathbf{n})$: Heuristic estimated cost from node $\mathbf{n}$ to goal $\mathbf{g}$.
- $f(\mathbf{n})$: Total estimated cost function guiding the search.

The algorithm iteratively explores neighboring nodes until the optimal path from $\mathbf{s}$ to $\mathbf{g}$ is found or all possibilities are exhausted.

Notes: This theoretical model assumes:

- The joint configuration space is discretized into uniform intervals.
- The space has a toroidal structure, allowing joints to wrap around from $-\pi$ to π.
- Each step between neighboring configurations has an equal cost of 1.
- The heuristic is admissible (never overestimates cost), ensuring optimality of the solution.

Source: Adapted from standard A* algorithm definitions:
https://en.wikipedia.org/wiki/A*_search_algorithm

Ref. By: GD1

Preconditions for TM:AStarJointSpace: The start and goal nodes must be within the defined grid dimensions.

Derivation for TM:AStarJointSpace: 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	**A* Heuristic in Discretized Toroidal Joint Space**
SI Units	Unitless (grid indices)
Equation	$$h(\mathbf{n}) = \sum_{i=1}^{n} \min\left(\lvert n_i - g_i \rvert, M - \lvert n_i - g_i \rvert\right)$$
Description	This equation defines the toroidal Manhattan distance used as the heuristic function $h(\mathbf{n})$ in the A* algorithm for robotic arm joint-space planning. n is the number of joints (dimensions of the search space). $\mathbf{n} = [n_1, n_2, \ldots, n_n]$ is the current node index in the discretized joint space. $\mathbf{g} = [g_1, g_2, \ldots, g_n]$ is the goal node index. M is the number of discrete bins per joint (grid resolution). The min term handles wrap-around behavior in toroidal space. For example, the distance from bin 0 to bin $M-1$ is 1, not $M-1$. Assumptions: • The joint space is discretized uniformly into M bins per joint. • The joint angle space is toroidal (wraps around at $-\pi$ and π). • The heuristic must be admissible and consistent for A* to be optimal.
Source	Adapted from: https://en.wikipedia.org/wiki/A*_search_algorithm
Ref. By	GD4.2.2

Detailed derivation of A* heuristic in toroidal joint space

The A* algorithm requires a heuristic function $h(\mathbf{n})$ to estimate the cost from a current node $\mathbf{n}$ to the goal node $\mathbf{g}$. In a discretized n-dimensional joint space with M bins per dimension (corresponding to uniformly sampled angles), we aim to define a heuristic that:

- Is admissible (never overestimates true cost),
- Is consistent (ensures optimal pathfinding),
- Accounts for toroidal topology (wrap-around effect).

1. In each joint dimension i, the shortest distance from current index n_i to goal index g_i must consider both clockwise and counter-clockwise directions due to wrapping.

2. The forward (absolute) distance is $|n_i - g_i|$.

3. The wrap-around distance is $M - |n_i - g_i|$.

4. The effective shortest distance in that dimension is:

$$d_i = \min(|n_i - g_i|, M - |n_i - g_i|)$$

5. Summing over all n joint dimensions gives the total heuristic:

$$h(\mathbf{n}) = \sum_{i=1}^{n} d_i = \sum_{i=1}^{n} \min(|n_i - g_i|, M - |n_i - g_i|)$$

This formulation respects the periodicity of angular joint space, making it ideal for pathfinding on robotic arms with revolute joints. It guarantees optimality and efficiency in A* search when used as the heuristic function.

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	**End-Effector Position Definition**
Symbol	$\mathbf{x} = (x, y)$
SI Units	m
Equation	$$x = \sum_{i=1}^{n} L_i \cos\left(\sum_{j=1}^{i} \theta_j\right), \quad y = \sum_{i=1}^{n} L_i \sin\left(\sum_{j=1}^{i} \theta_j\right)$$
Description	This definition gives the Cartesian coordinates (x, y) of the end-effector, computed from the joint angles θ_i and link lengths L_i of a planar n-link robotic manipulator. • θ_i: Joint angle (radians) • L_i: Length of link i (mm) • $\mathbf{x} = (x, y)$: Cartesian coordinates of the end-effector This data definition specifies how to calculate $\mathbf{x}$ and provides a consistent notation for use in instance models such as path planning and obstacle avoidance.
Source	Robotics textbook or GD:ForwardKinematics
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	Formal Definition and Description
`Angle`	$\theta \in [-\pi, \pi] \subset \mathbb{R}$ — Represents joint angles of the robotic arm in radians.
`Length`	$L \in \mathbb{R}_{\geq 0}$ — Represents physical lengths (e.g., arm links, obstacle radii) in millimetres.
`Position`	$\mathbf{x} = (x, y) \in \mathbb{R}^2$ — A 2D Cartesian coordinate in millimetres.
`Velocity`	$\dot{\theta} \in \mathbb{R}$ — Joint angular velocity in radians per second.
`Acceleration`	$\ddot{\theta} \in \mathbb{R}$ — Joint angular acceleration in radians per second squared.
`Obstacle`	$O = (x, y, r)$ with $(x, y) \in \mathbb{R}^2$, $r \in \mathbb{R}_{>0}$ — Represents a circular obstacle.
`Boolean`	$b \in \{\texttt{true}, \texttt{false}\}$ — A binary flag indicating feasibility of a path or result.

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:

Number	IM1
Label	**2D Robotic Arm Path Planning**
Input	• $\mathbf{q}_{\text{init}}, \mathbf{q}_{\text{goal}}$: Initial and goal joint configurations. • $O_i = (x_i, y_i, r_i)$: Circular obstacles in Cartesian space. • Joint limits (A3) and other constraints.
Output	A collision-free path (or empty if infeasible), represented as $$\mathbf{Q}_{\text{path}} = [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_N]$$ each $\mathbf{q}_k$ a valid joint configuration. Optionally, a continuous trajectory $\mathbf{q}(t)$ can be generated through interpolation.
Description	This instance model unifies path planning and collision avoidance for a 2D robotic arm with n revolute joints. **Key steps:** 1. **Discrete Search (A*):** Use the toroidal joint-space A* model (TM4.2.2) to find a minimal-step path of valid configurations from $\mathbf{q}_{\text{init}}$ to $\mathbf{q}_{\text{goal}}$. 2. **Collision Detection:** Each candidate $\mathbf{q}$ is validated via segment-circle checks (TM4.2.2), using forward kinematics (TM4.2.2) to map joint angles to link endpoints. 3. **Trajectory Generation (Optional):** If a discrete path is found, a continuous trajectory $\mathbf{q}(t)$ can be formed (e.g., spline interpolation) for smooth execution. The path or trajectory must satisfy: • No collisions with obstacles (A5). • Joint limits do not exceed physical constraints (A3). • If no valid path, return an infeasibility result. This addresses the goal statements of computing a collision-free path (GS1) and minimizing the path length (via the A* step count) (GS2).
Sources	TM4.2.2, TM4.2.2, TM4.2.2
Ref. By	R3, R4

Derivation of 2D Robotic Arm Path Planning

This instance model is derived by combining the following theoretical models and assumptions:

- **Forward Kinematics** (TM4.2.2): Maps any joint configuration $\mathbf{q}$ to the Cartesian positions of the arm's links, enabling collision checks.
- **Collision Detection** (TM4.2.2): Determines whether any link segment intersects a circular obstacle O_i.
- **Toroidal A* Search** (TM4.2.2): Provides a pathfinding strategy in a discretized joint-space grid with wrap-around angles, applying an admissible heuristic to ensure an optimal solution with minimal steps.

At each step, the A* algorithm proposes a neighboring joint configuration, which is validated for collisions using the segment-circle model. If feasible, the configuration is explored in the search process. The final sequence of configurations from $\mathbf{q}_{\text{init}}$ to $\mathbf{q}_{\text{goal}}$ constitutes the minimal-step, collision-free path. Optional trajectory generation interpolates these discrete points into a continuous function $\mathbf{q}(t)$, respecting joint limits (A3) and ensuring a smooth motion plan.

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.

(*) 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 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%

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

As previously discussed, a correct trajectory must be collision-free, kinematically feasible, and respect joint and workspace constraints.

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, collectively defined as the input set $\mathcal{I}$:

$$\mathcal{I} = \{\mathbf{q}_{\text{init}}, \mathbf{x}_{\text{goal}}, \mathcal{O}, \mathbf{L}\}$$

where:

- $\mathbf{q}_{\text{init}} = (\theta_1, \ldots, \theta_N)$: initial joint angles (radians).

- $\mathbf{x}_{\text{goal}} = (x, y)$: goal position in Cartesian coordinates (m).
- $\mathcal{O} = \{(x_i, y_i, r_i) \mid i = 1, \ldots, M\}$: obstacle positions and radii (m).
- $\mathbf{L} = (L_1, \ldots, L_N)$: link lengths of the robotic arm (m).

Trace to: IM1

R2: The system shall echo all elements in $\mathcal{I}$ to confirm correctness before computation.

R3: The system shall perform the following calculations:

- Use forward kinematics (TM4.2.2) to compute end-effector positions from $\mathbf{q}$.
- Use A* search (TM4.2.2) to generate a collision-free path in joint space from $\mathbf{q}_{\text{init}}$ to the joint configuration corresponding to $\mathbf{x}_{\text{goal}}$.
- Construct a time-indexed path $\mathbf{q}(t)$ through intermediate joint configurations for smooth execution.

R4: The system shall verify that:

- The path $\mathbf{q}(t)$ is collision-free with respect to $\mathcal{O}$ (TM4.2.2).
- The final configuration $\mathbf{q}_{\text{final}}$ maps via forward kinematics to the goal position $\mathbf{x}_{\text{goal}}$ (TM4.2.2).
- The trajectory is continuous, feasible, and respects all joint and link constraints.

R5: The system shall output the following:

- A discrete sequence $\mathbf{q}(t_0), \mathbf{q}(t_1), \ldots, \mathbf{q}(t_k)$ in joint space.
- Corresponding Cartesian path $\mathbf{x}(t)$ of the end-effector.
- A visualization including the arm trajectory 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 2D 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.

Accuracy is verified through both analytical comparison and visual confirmation:

– **Analytical oracle:** For each configuration along the planned trajectory $\mathbf{q}(t)$, the system computes the expected end-effector position $\mathbf{x}_{\text{expected}}(t)$ using the forward kinematics model (TM4.2.2). The output is deemed accurate if:

$$\|\mathbf{x}_{\text{actual}}(t) - \mathbf{x}_{\text{expected}}(t)\| < 0.001 \text{ m}$$

– **Visual oracle:** The GUI module displays the animated arm trajectory alongside obstacles. Users can visually inspect whether the motion respects the environment constraints and achieves the goal position with expected smoothness and precision.

See V&V plan for test design and validation procedures.

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. V&V plan The outputs shall be compared against theoretical models and experimental data.

NFR4: **Understandability**
The software architecture shall be modular and self-descriptive. Each module shall include a clearly written interface specification and a corresponding documentation file. Every public function shall include docstrings that specify input/output types and purpose. Inline comments shall be added to explain non-obvious implementation logic. Understandability will be evaluated through peer review and code inspection checklists.

NFR5: **Reusability**
The system shall support reuse of key planning components (e.g., collision checking, A* search, forward kinematics) across different robotic manipulators. All configuration-dependent parameters (e.g., number of joints, link lengths) shall be externally specified in JSON or parameter files. Module interfaces shall not be hardcoded to any specific robot model, enabling reuse in future planar path-planning 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 (m): $L_1 = 50$, $L_2 = 40$
  - Joint angle range (rad): $[-\pi, \pi]$
  - Obstacle radii (m): $r_i \in [5, 15]$

  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 6 shows the dependencies of theoretical models, general definitions, data definitions, and instance models with each other. Table 7 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.

	TM1	TM2	TM3	GD1	DD1	IM1
TM2						
TM1			X			
TM2						
TM3						
GD1	X					
DD1				X		
IM1				X		

Table 6: 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

	IM1	DC	R1	R2	R3	R4	R5
IM1		X				X	X
R1							
R2						X	
R3					X		
R4	X						
R5	X	X				X	X

Table 7: Traceability Matrix Showing the Connections Between Requirements and Instance Models

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.

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.

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

Table 8: Values of Auxiliary Constants

Appendix — Reflection

[Not required for CAS 741 —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. 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."

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?