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

[This template is intended for use by CAS 741. For CAS 741 the template should be used exactly as given, except the Reflection Appendix can be deleted. For the capstone course it is a source of ideas, but shouldn't be followed exactly. The exception is the reflection appendix. All capstone SRS documents should have a reflection appendix. —TPLT]

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
cm	length	centimetre
s	time	second
q	joint angle	radians
$\dot{q}$	joint velocity	rad/s
$\ddot{q}$	joint acceleration	rad/s^2

[Only include the units that your SRS actually uses. —TPLT]

[Derived units, like newtons, pascal, etc, should show their derivation (the units they are derived from) if their constituent units are in the table of units (that is, if the units they are derived from are used in the document). For instance, the derivation of pascals as Pa = N m^{-2} is shown if newtons and m are both in the table. The derivations of newtons would not be shown if kg and s are not both in the table. —TPLT]

[The symbol for units named after people use capital letters, but the name of the unit itself uses lower case. For instance, pascals use the symbol Pa, watts use the symbol W, teslas use the symbol T, newtons use the symbol N, etc. The one exception to this is degree Celsius. Details on writing metric units can be found on the NIST web-page. —TPLT]

1.2 Table of Symbols

The table that follows summarizes the symbols used in this document along with their units. The choice of symbols was made to be consistent with the heat transfer literature and with existing documentation for solar water heating systems. The symbols are listed in alphabetical order.

symbol	unit	description
L	cm	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	cm	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	cm	Position of the i-th obstacle (circular)
r_i	cm	Radius of the i-th obstacle
M	-	Number of discrete joint positions in A* search grid
θ_i	radians	Discretized joint angle for A* search
$\mathcal{G}$	-	Occupancy grid (joint space)
$\mathbf{p}_i$	cm	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

[Add any other abbreviations or acronyms that you add —TPLT]

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.

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.

Figure 1: System Context

  - 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.

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 two revolute joints, defining a two-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.

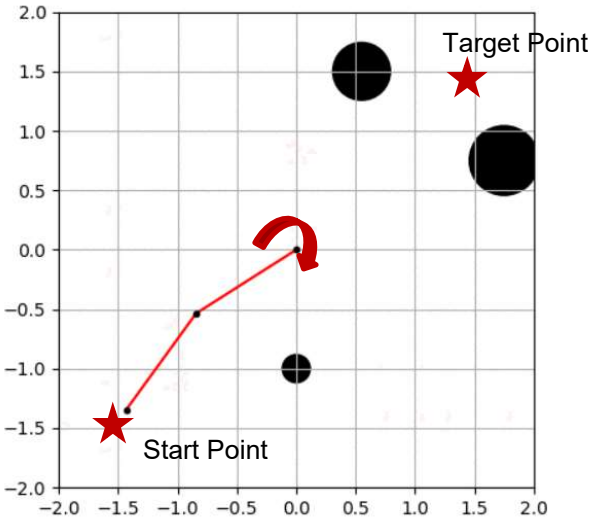


Figure 2: Robotic arm finding a path to target point

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.

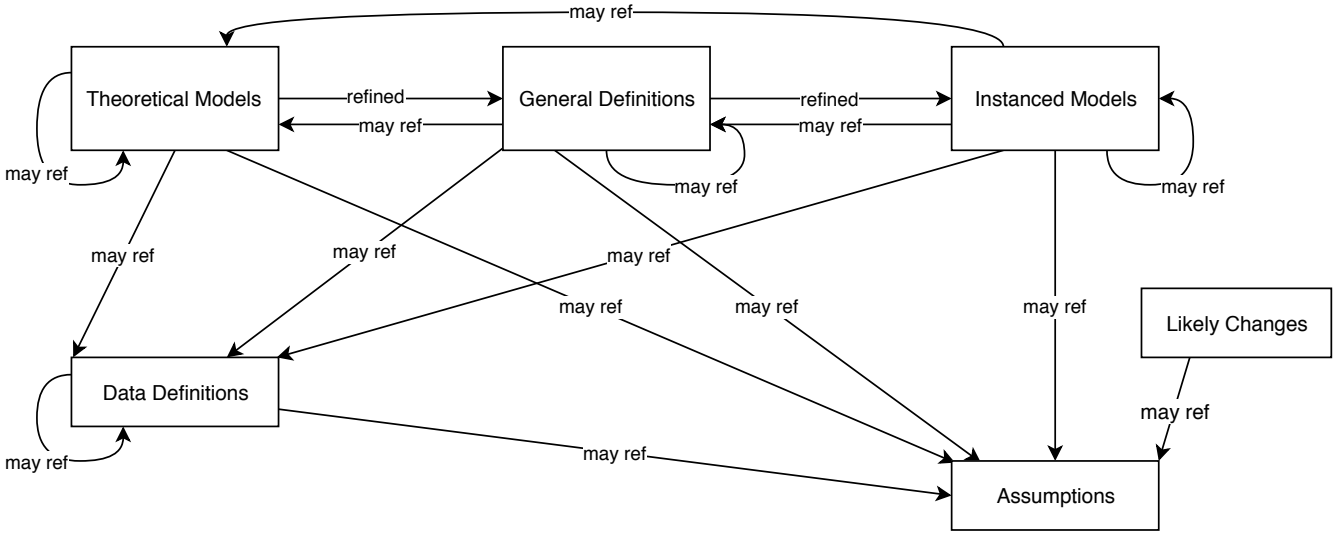
GS2: Minimize the total path length while ensuring obstacle avoidance.

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 (cm).
- L_2: Length of the second link (cm).

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

["Ref. By" is used repeatedly with the different types of information. This stands for Referenced By. It means that the models, definitions and assumptions listed reference the current model, definition or assumption. This information is given for traceability. Ref. By provides a pointer in the opposite direction to what we commonly do. You still need to have a reference in the other direction pointing to the current model, definition or assumption. As an example, if TM1 is referenced by GD2, that means that GD2 will explicitly include a reference to TM1. —TPLT]

4.2.3 General Definitions

[General Definitions (GDs) are a refinement of one or more TMs, and/or of other GDs. The GDs are less abstract than the TMs. Generally the reduction in abstraction is possible through invoking (using/referencing) Assumptions. For instance, the TM could be Newton's Law of Cooling stated abstracting. The GD could take the general law and apply it to get a 1D equation. —TPLT]

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

[Some projects may not have any content for this section, but the section heading should be kept. —TPLT] [Modify the examples below for your problem, and add additional definitions as appropriate. —TPLT]

Number	GD1
Label	**Newton's law of cooling**
SI Units	$\mathrm{W\,m^{-2}}$
Equation	$q(t) = h\Delta T(t)$
Description	Newton's law of cooling describes convective cooling from a surface. The law is stated as: the rate of heat loss from a body is proportional to the difference in temperatures between the body and its surroundings. $q(t)$ is the thermal flux ($\mathrm{W\,m^{-2}}$). h is the heat transfer coefficient, assumed independent of T (A**??**) ($\mathrm{W\,m^{-2}\,{}^{\circ}C^{-1}}$). $\Delta T(t) = T(t) - T_{\mathrm{env}}(t)$ is the time-dependent thermal gradient between the environment and the object (°C).
Source	Citation here
Ref. By	DD1, DD**??**

Detailed derivation of simplified rate of change of temperature

[This may be necessary when the necessary information does not fit in the description field. —TPLT] [Derivations are important for justifying a given GD. You want it to be clear where the equation came from. —TPLT]

4.2.4 Data Definitions

[The Data Definitions are definitions of symbols and equations that are given for the problem. They are not derived; they are simply used by other models. For instance, if a problem depends on density, there may be a data definition for the equation defining density. The DDs are given information that you can use in your other modules. —TPLT]

[All Data Definitions should be used (referenced) by at least one other model. —TPLT]

This section collects and defines all the data needed to build the instance models. The dimension of each quantity is also given. [Modify the examples below for your problem, and add additional definitions as appropriate. —TPLT]

Number	DD1
Label	**Heat flux out of coil**
Symbol	q_C
SI Units	$\mathrm{W\,m^{-2}}$
Equation	$q_C(t) = h_C(T_C - T_W(t))$, over area A_C
Description	T_C is the temperature of the coil (°C). T_W is the temperature of the water (°C). The heat flux out of the coil, q_C ($\mathrm{W\,m^{-2}}$), is found by assuming that Newton's Law of Cooling applies (A**??**). This law (GD1) is used on the surface of the coil, which has area A_C ($\mathrm{m^2}$) and heat transfer coefficient h_C ($\mathrm{W\,m^{-2}\,°C^{-1}}$). This equation assumes that the temperature of the coil is constant over time (A**??**) and that it does not vary along the length of the coil (A**??**).
Sources	Citation here
Ref. By	IM1

4.2.5 Data Types

[This section is optional. In many scientific computing programs it isn't necessary, since the inputs and outpus are straightforward types, like reals, integers, and sequences of reals and integers. However, for some problems it is very helpful to capture the type information. —TPLT]

[The data types are not derived; they are simply stated and used by other models. —TPLT]

[All data types must be used by at least one of the models. —TPLT]

[For the mathematical notation for expressing types, the recommendation is to use the notation of Hoffman and Strooper (1995). —TPLT]

This section collects and defines all the data types needed to document the models. [Modify the examples below for your problem, and add additional definitions as appropriate. —TPLT]

Type Name	Name for Type
Type Def	mathematical definition of the type
Description	description here
Sources	Citation here, if the type is borrowed from another source

4.2.6 Instance Models

[The motivation for this section is to reduce the problem defined in "Physical System Description" (Section 4.1.2) to one expressed in mathematical terms. The IMs are built by refining the TMs and/or GDs. This section should remain abstract. The SRS should specify the requirements without considering the implementation. —TPLT]

This section transforms the problem defined in Section 4.1 into one which is 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 goals [reference your goals —TPLT] are solved by [reference your instance models —TPLT]. [other details, with cross-references where appropriate. —TPLT] [Modify the examples below for your problem, and add additional models as appropriate. —TPLT]

Number	IM1
Label	**Energy balance on water to find T_W**
Input	m_W, C_W, h_C, A_C, h_P, A_P, t_{final}, T_C, T_{init}, $T_P(t)$ from IM**??** The input is constrained so that $T_{\text{init}} \leq T_C$ (A**??**)
Output	$T_W(t)$, $0 \leq t \leq t_{\text{final}}$, such that $\frac{dT_W}{dt} = \frac{1}{\tau_W}[(T_C - T_W(t)) + \eta(T_P(t) - T_W(t))]$, $T_W(0) = T_P(0) = T_{\text{init}}$ (A**??**) and $T_P(t)$ from IM**??**
Description	T_W is the water temperature (°C). T_P is the PCM temperature (°C). T_C is the coil temperature (°C). $\tau_W = \frac{m_W C_W}{h_C A_C}$ is a constant (s). $\eta = \frac{h_P A_P}{h_C A_C}$ is a constant (dimensionless). The above equation applies as long as the water is in liquid form, $0 < T_W < 100^o$C, where 0^oC and 100^oC are the melting and boiling points of water, respectively (A**??**, A**??**).
Sources	Citation here
Ref. By	IM**??**

Derivation of ...

[The derivation shows how the IM is derived from the TMs/GDs. In cases where the derivation cannot be described under the Description field, it will be necessary to include this subsection. —TPLT]

4.2.7 Input Data Constraints

Table 2 shows the data constraints on the input output 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. The software constraints will be helpful in the design stage for picking suitable algorithms. The constraints are conservative, to give the user of the model the flexibility to experiment with unusual situations. The column of typical values is intended to provide a feel for a common scenario. The uncertainty column provides an estimate of the confidence with which the physical quantities can be measured. This information would be part of the input if one were performing an uncertainty quantification exercise.

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

Table 2: Input Variables

Var	Physical Constraints	Software Constraints	Typical Value	Uncertainty
L	$L > 0$	$L_{\min} \leq L \leq L_{\max}$	1.5 m	10%

(*) [you might need to add some notes or clarifications —TPLT]

Table 4: Specification Parameter Values

Var	Value
$L_{\min}$	0.1 m

4.2.8 Properties of a Correct Solution

A correct solution must exhibit [fill in the details —TPLT]. [These properties are in addition to the stated requirements. There is no need to repeat the requirements here. These additional properties may not exist for every problem. Examples include conservation laws

Table 6: Output Variables

Var	Physical Constraints
T_W	$T_{\text{init}} \leq T_W \leq T_C$ (by A??)

(like conservation of energy or mass) and known constraints on outputs, which are usually summarized in tabular form. A sample table is shown in Table 6 —TPLT]

[This section is not for test cases or techniques for verification and validation. Those topics will be addressed in the Verification and Validation plan. —TPLT]

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 (cm).
- Obstacle positions $O_i = (x_i, y_i)$ and radii r_i (cm).
- Link lengths L_1, L_2 of the robotic arm (cm).

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.

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

[Provide a rationale for the decisions made in the documentation. Rationale should be provided for scope decisions, modelling decisions, assumptions and typical values. —TPLT]

6 Likely Changes

LC1: [Give the likely changes, with a reference to the related assumption (aref), as appropriate. —TPLT]

7 Unlikely Changes

LC2: [Give the unlikely changes. The design can assume that the changes listed will not occur. —TPLT]

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 10 shows the dependencies of theoretical models, general definitions, data definitions, instance models, and likely changes on the assumptions.

[You will have to modify these tables for your problem. —TPLT]

[The traceability matrix is not generally symmetric. If GD1 uses A1, that means that GD1's derivation or presentation requires invocation of A1. A1 does not use GD1. A1 is "used by" GD1. —TPLT]

[The traceability matrix is challenging to maintain manually. Please do your best. In the future tools (like Drasil) will make this much easier. —TPLT]

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.

9 Development Plan

[This section is optional. It is used to explain the plan for developing the software. In particular, this section gives a list of the order in which the requirements will be implemented.

	TM??	TM??	TM??	GD1	GD??	DD1	DD??	DD??	DD??	IM1	IM??	IM??
TM??												
TM??			X									
TM??												
GD1												
GD??	X											
DD1				X								
DD??				X								
DD??												
DD??								X				
IM1					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

	IM1	IM??	IM??	IM??	4.2.7	R??	R??
IM1		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??
TM??	X																		
TM??																			
TM??																			
GD1		X																	
GD??			X	X	X	X													
DD1							X	X	X										
DD??			X	X						X									
DD??																			
DD??																			
IM1											X	X		X	X	X			X
IM??												X	X			X	X	X	
IM??														X					X
IM??													X					X	
LC??				X															
LC??								X											
LC??									X										
LC??											X								
LC??												X							
LC??															X				

Table 10: Traceability Matrix Showing the Connections Between Assumptions and Other Items

In the context of a course this is where you can indicate which requirements will be implemented as part of the course, and which will be "faked" as future work. This section can be organized as a prioritized list of requirements, or it could should the requirements that will be implemented for "phase 1", "phase 2", etc. —TPLT]

10 Values of Auxiliary Constants

[Show the values of the symbolic parameters introduced in the report. —TPLT]

[The definition of the requirements will likely call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance. —TPLT]

[The value of FRACTION, for the Maintainability NFR would be given here. —TPLT]

References

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

[The following is not part of the template, just some things to consider when filing in the template. —TPLT]
[Grammar, flow and LaTeXadvice:

- For Mac users `*.DS_Store` should be in `.gitignore`
- LaTeX and formatting rules
  - Variables are italic, everything else not, includes subscripts (link to document)
    - Conventions
    - Watch out for implied multiplication
  - Use BibTeX
  - Use cross-referencing
- Grammar and writing rules
  - Acronyms expanded on first usage (not just in table of acronyms)
  - "In order to" should be "to"

—TPLT]
[Advice on using the template:

- Difference between physical and software constraints
- Properties of a correct solution means *additional* properties, not a restating of the requirements (may be "not applicable" for your problem). If you have a table of output constraints, then these are properties of a correct solution.
- Assumptions have to be invoked somewhere
- "Referenced by" implies that there is an explicit reference
- Think of traceability matrix, list of assumption invocations and list of reference by fields as automatically generatable
- If you say the format of the output (plot, table etc), then your requirement could be more abstract

—TPLT]

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?