**ENPM808X: Software Development for Robotics**

Fall '22

# Mid-Term Project Proposal

Inverse Kinematics Solver for KUKA Arm using Software Development Methodologies

## Team Members

**Adarsh Malapaka** - 118119625

**Kumara Ritvik Oruganti** - 117368963

**Master of Engineering Program (Robotics)**
**A. James Clark School of Engineering**
**University of Maryland, College Park, MD - 20742**

## Contents

## 1 Overview

Acme Robotics, Inc. is venturing into developing robotic processes solutions for automating assembly line production of automobiles. In this project, we propose to develop a manipulator-based solution to transport various components from one stage of the assembly line to another. This shall result in a reduction in the inaccuracies induced by human intervention, thereby increasing the productivity of the plant. To this extent, the 7-DOF robot, KUKA LBR IIWA 14 R820 CR has been chosen for two primary reasons namely: 1. the extended workspace of the robot near the base enables picking and placing of a multitude of objects with varied geometric and inertial properties, 2. the usage of an existing industrial manipulator provides the company with on-demand and round-the-clock inspection and maintenance in the event of malfunction, by the manufacturer of the robot manipulator.

This software assumes that there are no defects in the mechanical and electronic components of the robot. The link lengths are assumed to be perfect lengths without any offset due to external factors, as given in the datasheet of the manipulator.

## 2 Process

The software design and development process shall involve adhering to an amalgamation of Agile Iterative Processes and pair-programming techniques over a course of two weeks. The efficacy of the implementation shall be verified using the test-driven development approach. The driver and navigator roles are exchanged during each phase of the project to effectively develop and complete the backlogs. The various components of the software implementation are listed below:

- **Kinematics Solver**: This shall contains both forward position kinematics and inverse velocity kinematics solvers.
- **Controller**: In order to reduce the error in the trajectory followed by the robot, a controller is required.
- **Simulator**: To visualize the robot before deploying it on actual hardware as a means of verifying solution.

### 2.1 Technology Stack

We shall use C++'s Object Oriented programming capabilities to make the software scalable and adaptable with Visual Studio Code IDE configured with clangd, cppcheck, valgrind and cpplint.

#### 2.1.1 Libraries

The software might include 1. NumCpp 2. KDL 3. Math 4. Eigen 5. Matplotplusplus or Peter Corke's Robotics Toolbox Visualization libraries.

#### 2.1.2 Build System

CMake is used as a build system to provide cross platform compilation capability for the software.

### 2.2 Algorithm

#### 2.2.1 Forward Kinematics

Forward kinematics is calculated by using the standard form of the Denavit-Hartenberg Convention (Spong's Convention) to assign coordinate frames to the manipulator and computing the homogeneous transformation matrices to obtain the end-effector pose of the manipulator.

### 2.2.2 Inverse Kinematics

Inverse Kinematics is used to get the joint angles of the manipulator for a given cartesian goal pose in the feasible workspace, using the velocity approach. It involves the sampling based path generation to get intermediate joint configurations to reach the goal. If X is the column vector containing end-effector cartesian velocities, 'q' is the column vector containing joint angles, and J is the Jacobian matrix:

$$\dot{q} = J^{-1} * \dot{X} \tag{1}$$

Using numerical integration to obtain the joint angles,

$$q_{new} = q_{prev} + \dot{q} * \delta t \tag{2}$$

### 2.2.3 Control System

A PID based controller is used as a parameter to sample the generated path to get intermediate joint state configurations.

## 3 Risks and Mitigation

- The possible risks include singularity configurations of the robotic manipulator in the generated path. There can be multiple joint configurations for the given pose. The robot may not follow a smooth position and joint velocity trajectory and kinematics solutions can be affected by mechanical offsets, hardware and manufacturing errors in the links of the manipulator.
- Mitigations include checking the desired pose in the robot's reachable workspace and joint angles' limitations. Saturation blocks will be cascaded with the PID Controller output to eliminate erroneous values. These will be rigorously tested as a part of the Google Unit Test Framework.

## 4 Final Deliverables

The deliverable will be a cross platform compilable software with appropriate functionalities and visualization for the generation of a path between the given initial configuration and final goal configuration.