Manipulator arm path planner (IK solver)

An IK solver and path planner for Acme Robotics

Kavyashree Devadiga
117398045

Aswath Muthuselvam
118286204

The proposed system will provide a solution for a 6DOF robotic arm path planning consisting of an IK solver and the output will be simulated in a virtual environment.

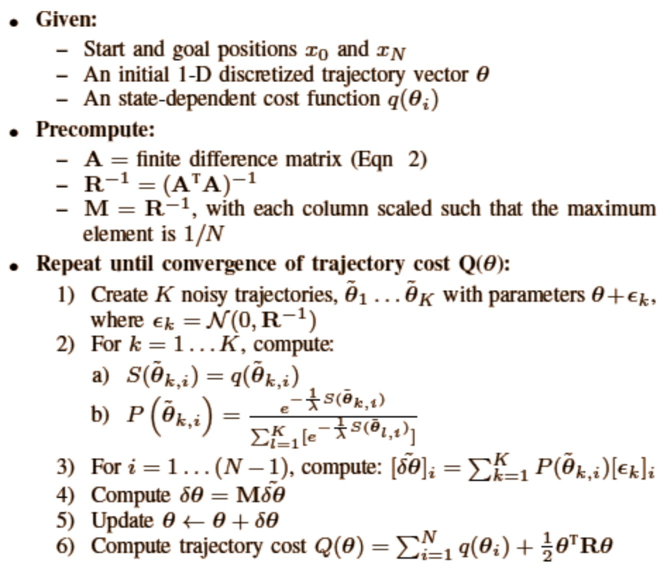
Our system will provide a low dependency implementation for the arm manipulator path planning. The input to the system will be the desired goal cartesian coordinates. This input will be applied to a 6 DOF UR5 arm with set initial points and known configuration. Based on this data the IK solver will calculate the configuration of individual arm components. To avoid unforeseen problems, each calculation will be validated against the arm constraints.

There are a lot of libraries involved in building a software program. In our case, we are utilizing freely available and widely used open-source libraries. The Eigen C++ library is used for matrix calculations. Eigen supports all standard numerics and is very clean, reliable, and easy to use. Using Eigen we can avoid unnecessary matrix code implementation and can focus on the main arm kinematics logic. Eigen is free software with MPL2 (Mozilla Public License), which is a simple copyleft license. MoveIt is an open-source robotics software package with a compilation of various algorithms for path and trajectory planning and inverse kinematics solvers. Moveit is licensed under BSDv3 (Berkeley Software Distribution). OMPL (Open Motion Planning Library) is another robotic software package that contains various state-of-the-art sampling-based motion planning algorithms. The OMPL software package is maintained by the Kavraki Lab at Rice University. OMPL is licensed under a 3-clause BSD license. Coppelia Sim is a simulation software that enables robotic arm-based simulations. Various robotic arms that are used in industry are available to experiment. Coppelia sim offers various licenses and we are using the educational license.

The technical ideas of our implementation include Inverse kinematics, Path planning, and sending control messages. The output of the IK solver will be fed as an input to the path planner module. This module will focus on arm trajectory planning. There are various sampling-based planners, we are using RRT Star.

Stochastic Trajectory Optimization for Motion Planning (STOMP) will be used for path planning for moving the robotic arm to a goal state. STOMP generates noisy trajectories and then optimizes it based on cost function that includes obstacle and smoothness. Gradients are not used to optimize the path, thus making this algorithm faster than the ones that use gradients.

will import methods for this from MoveIt libraries. The steps involved in STOMP are as shown in figure 1.



```
• Given:
    – Start and goal positions $x_0$ and $x_N$
    – An initial 1-D discretized trajectory vector $\theta$
    – An state-dependent cost function $q(\theta_i)$
• Precompute:
    – $\mathbf{A} =$ finite difference matrix (Eqn 2)
    – $\mathbf{R}^{-1} = (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}$
    – $\mathbf{M} = \mathbf{R}^{-1}$, with each column scaled such that the maximum
      element is $1/N$
• Repeat until convergence of trajectory cost Q($\theta$):
    1) Create $K$ noisy trajectories, $\tilde{\theta}_1 \ldots \tilde{\theta}_K$ with parameters $\theta + \epsilon_k$,
       where $\epsilon_k = \mathcal{N}(0, \mathbf{R}^{-1})$
    2) For $k = 1 \ldots K$, compute:
        a) $S(\tilde{\theta}_{k,i}) = q(\tilde{\theta}_{k,i})$
        b) $P\left(\tilde{\theta}_{k,i}\right) = \frac{e^{-\frac{1}{\lambda}S(\tilde{\theta}_{k,i})}}{\sum_{l=1}^{K}[e^{-\frac{1}{\lambda}S(\tilde{\theta}_{l,i})}]}$
    3) For $i = 1 \ldots (N-1)$, compute: $[\tilde{\delta\theta}]_i = \sum_{k=1}^{K} P(\tilde{\theta}_{k,i})[\epsilon_k]_i$
    4) Compute $\delta\theta = \mathbf{M}\tilde{\delta\theta}$
    5) Update $\theta \leftarrow \theta + \delta\theta$
    6) Compute trajectory cost $Q(\theta) = \sum_{i=1}^{N} q(\theta_i) + \frac{1}{2}\theta^{\mathrm{T}}\mathbf{R}\theta$
```

Figure 1. STOMP Algorithm, referred from here [1]

A build system is required to compile the written program and generate an executable, during this process, the build system locates dependencies like external libraries and uses it to generate the final executable. We are making use of CMake as our build system. It is widely used in industry and supports many platforms and generating configurations for various IDEs (Integrated Development Environment). CMake compiles C/C++ codes and generates "out of source" build files.

After integrating into the real world the system has some potential risks. The time sampling for computing the dynamics of the robotic arm is large and not intended for direct use in the real world. While testing the system in the real world, there are a lot of noises involved, these include electrical noise like Electromagnetic disturbances in motor encoders, mechanical noise like frictional forces, and physical disturbances. Solution provided to this is that user can set error tolerance according to noise present in the system.

The calculations can get overwhelming and the development timeline might not match the proposed deadlines. In this case, the fallback plan would be - no velocity or acceleration considered system, just position calculations will be computed.

The modular approach of the project facilitates easy mitigation if there is an error in the output. After each micro-step, the constraint validation is done and the user is notified immediately about any error. We are also proposing info, debug and error logging during code execution. This system acceptability test will ensure the safety of the robotic arm and it will not perform any unwanted anomalous actions.

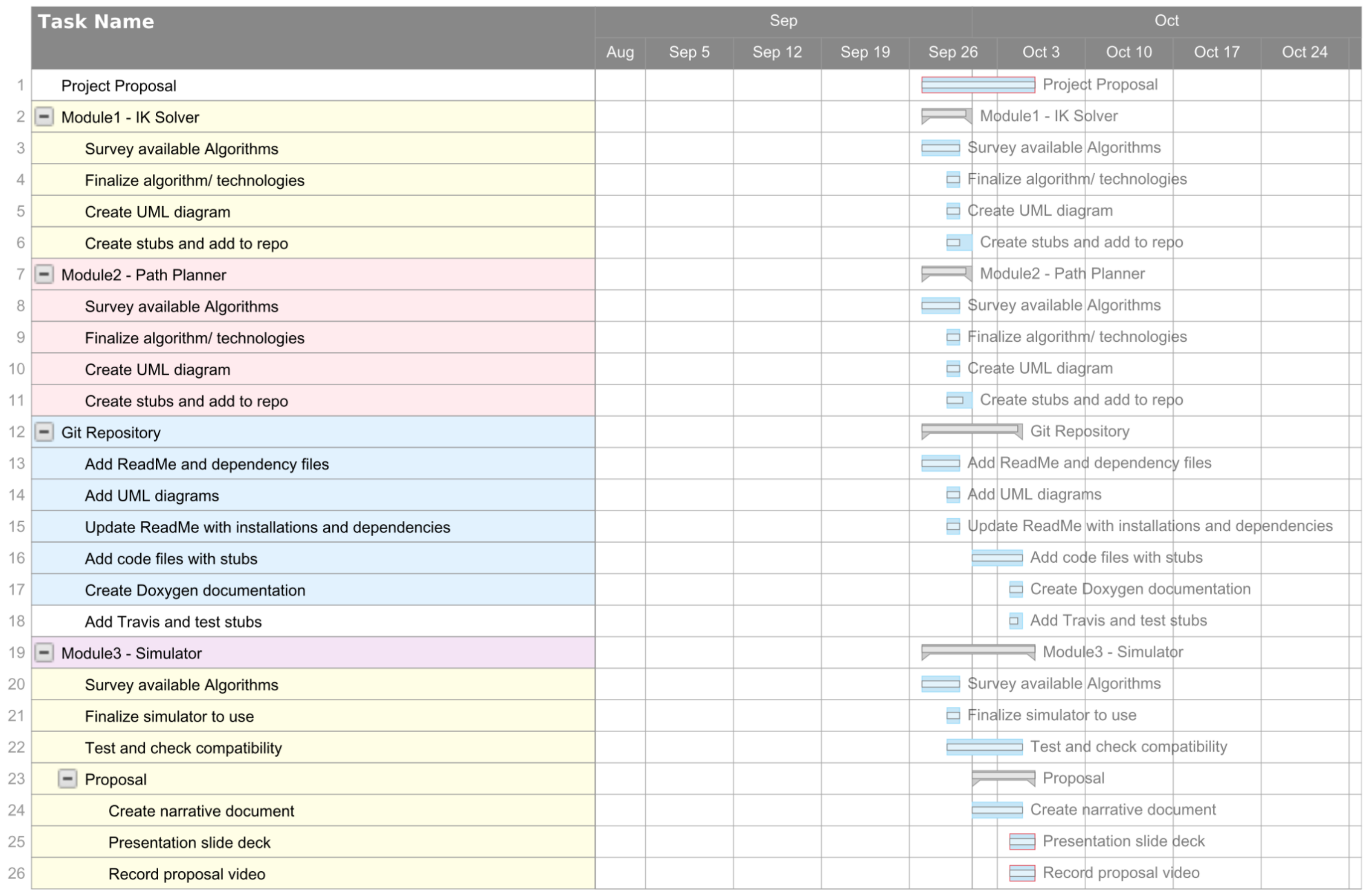
The development team of two will work together and use agile methodologies. A standard software pair programming will be implemented. The driver-navigator method will ensure code is inspected on the go and this collaboration will result in a bug-free, robust product. The driver and navigator will switch roles for different sub-module development.

We are proposing calculation cross correction before showing output. The system has a built-in forward dynamic solver for the generated results and compares it with user input. This in-system testing module ensures superior quality and improves project predictability.

The development period is three weeks. The deployment will be in a single release.

Citations:

[1] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," 2011 IEEE International Conference on Robotics and Automation, 2011, pp.4569-4574,doi: 10.1109/ICRA.2011.5980280.



	Task Name
1	Project Proposal
2	Module1 - IK Solver
3	Survey available Algorithms
4	Finalize algorithm/ technologies
5	Create UML diagram
6	Create stubs and add to repo
7	Module2 - Path Planner
8	Survey available Algorithms
9	Finalize algorithm/ technologies
10	Create UML diagram
11	Create stubs and add to repo
12	Git Repository
13	Add ReadMe and dependency files
14	Add UML diagrams
15	Update ReadMe with installations and dependencies
16	Add code files with stubs
17	Create Doxygen documentation
18	Add Travis and test stubs
19	Module3 - Simulator
20	Survey available Algorithms
21	Finalize simulator to use
22	Test and check compatibility
23	Proposal
24	Create narrative document
25	Presentation slide deck
26	Record proposal video

Figure.2 Project Backlog