MPC for Robotic Arm Path Planning and Control ECH-267 Final Project Report

Jonathan Dorsey *Member: No Sleep Club: est. 2017*
https://github.com/JonnyD1117/ECH-267-Adv.-Proc.-Control/tree/main/Project

***Abstract*—The objective of this project is to implement a simulated Path Planner and Controller using Model Predictive Control (MPC) to plan and control the behavior of a 2 degree of freedom (2DOF) robotic arm. The responsibility of the MPC planner will be to generate the 'optimal' path and to drive the arm from its current position to the next. This paper, uses the CasADi framework to implement, test, and identify the benefits and limitations of MPC for this robotic arm.**

***Index Terms*—Model Predictive Control, Robot Arm, Lagrange Equations, Path Planning, Obsticle Avoidance, DH Parameters, Trajectory Generation.**

I. Introduction

THE world of robotics is full of constraints, demands, and performance trade-offs that humans handle naturally on a daily basis. Unlike humans, robots require control and planning strategies which are flexibility enough to work around constraints and limitations, while accurately meeting control and performance objectives in uncertain environments. While the emergence of machine learning techniques and methods offers promise of stateoftheart improvements and performance, most robotic systems still demand a more practical and robust planning and control algorithms, capable of offering a compromise between the performance criteria, flexibility to navigate constraints, and amount of computational power required to compute valid control commands in real-time.

In the world of robot manipulators, tasks can range from relatively simple and coarse motions to extremely complex and detailed actions. One common control strategy which has seen great success, in robotics, in recent years is Model Predictive Control (MPC). MPC is an optimal control methodology which solves the a given optimal control problem (OCP) in a receding fashion, over a finite horizon. While these controllers are far more sophisticated than standard classical or modern control strategies, the increased complexity and computation can be applied to a wider class of control problems. This flexibility in natively handling constraints as well as naturally extending to nonlinear and multiple-input-multiple-output-systems makes MPC an attractive candidate control methodology for the vast world of robotics where tasks can range from autonomous mobile vehicles to robotic manipulators. While MPC has the obvious conns of requiring an approximate solution to an optimization problem, at each time step, the benefits which this methodology offers often make it a viable solution even with the added computationally expense.

While control is an important aspect of modern robotics, it is often more valuable to have an understanding of the intent or future actions which an autonomous system believes it should take, to accomplish a goal. In general, this problem is known as **path planning**, and is an important part of modern robotics research. Many modern path planning approaches use a vast array of different planning paradigms, such as discretization, graph, probablistic, and heuristic methods which have all shown great promise, and present there own unique benefits and limitations. Often in the more general case of motion planning, it is desirable to not only control the position, but also the velocity and acceleration of a system, in both space and time. However, for the purpose of this paper, only the more restricted case of path planning (e.g. position planning) will be considered.

To this point, another possible method for path planning is the use of MPC, as a optimal path planner. MPC has the potential to provide much of the same functionality as other planning strategies by implementing desireable behavior into a cost function or functional constraints. The ability to leverage and model predictive controller as a path planning with practically no changes to its implementation as a controller also offers an opportunity for systems using MPC to obtain some path planning capabilities for free.

While MPC has been received enormous amounts of research across many fields, including robotics, many of the applications in robotics focus MPC on mobile robotics such as drones and autonomous vehicles, with significantly less attention focused on the application of MPC as a controller or a planner for robotic manipulators.

II. Problem Statement

This paper will investigate the use of MPC as a general control scheme and its utility as a path planner for a simple two link planar robotic manipulator. By the use of simulation, this paper will develop and identify key elements in understanding some of the benefits and limitations of MPC in the context of articulated robotic manipulators.

A. Planar RR Robotic Arm

The manipulator used in this paper is a two link planar robot arm subject to gravity, fig.1, commonly known as a

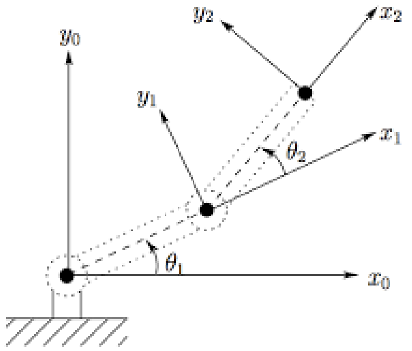


Fig. 1: RR Planar Manipulator subject to gravity

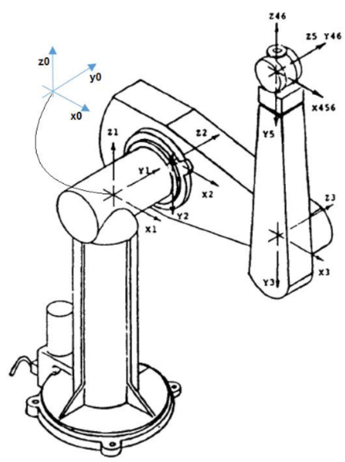


Fig. 2: Puma560 Robot with Joint Coordinate Frames

RR (revolute-revolute) robot. This model commonly does not include any rigid body dynamics and treats the robot as a system of point masses directly coupled with massless rods. While this configuration is very simple, it is conceptually simple, easy to derive its core dynamics, and facilitates a more focused study on the planning and control elements of the paper.

III. Background

To contextualize the implementation of MPC for on robotic manipulators, it is helpful to breakdown a few of the fundamental elements, constructs, and terminology which are commonly used in robotics literature.

A. Robotic Manipulators

In general robotic arms are classified as either prismatic (linear motion) or revolute (rotational motion) joints. The PUMA 560 [Fig.2], is an industry standard 6 degree of freedom (DOF) manipulator. It utilizes six revolute joints, connected in a serial fashion. This configuration of only rotating joints makes the PUMA 560 an **articulated robot**.

While the study of high DOF robots is well researched and would be present interesting and complicated scenarios for both control and planning, the scale and complexity of modeling the governing dynamic equations make robots like those in [Fig. 2] impractical for the scope and time restrictions of this paper. This paper, instead, opts to use the dynamically simple RR-robot described above, as its design and test platform for implementing MPC.

1) DH Parameters & Homogenous Transforms: The Denavit-Hartenberg (DH) parameters are an important tool in describing the geometry of any given robot, and prove to be highly effective in the formulation of joint transformations which enable a concise and universal method for transforming important quanties for kinematic and dynamic analysis of a robot.

The DH Parameters are defined to be...

$$\begin{aligned} a_i &= \text{ the distance from } \hat{Z}_i \text{ to } \hat{Z}_{i+1} \text{ measured along } \hat{X}_i \\ \alpha_i &= \text{ the angle from } \hat{Z}_i \text{ to } \hat{Z}_{i+1} \text{ measured about } \hat{X}_i \\ d_i &= \text{ the distance from } \hat{X}_{i-1} \text{ to } \hat{X}_i \text{ measured along } \hat{Z}_i; \text{ and} \\ \theta_i &= \text{ the angle from } \hat{X}_{i-1} \text{ to } \hat{X}_i \text{ measured about } \hat{Z}_i \end{aligned}$$

By utilizing the **homogeneous transform**, which describes the frame position and rotation required to define a vector from one frame to another, we generate a mapping of a vector to or from any coordinate system in the system.

$$\begin{bmatrix} {}^{A}P \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} & {}^{A}_{B}R & & {}^{A}P_{Borg} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} {}^{B}P \\ 1 \end{bmatrix} \tag{1}$$

$$P = {}^{A}_{B}T^{B}P \tag{2}$$

DH parameters can be used to define a single transformation from frame $\{i\}$ to frame $\{i-1\}$, by chaining explicit translations and rotations about specified axes. The matrices R and D represent the homogeneous transformation for rotation and translations repsectively, with the subscript of each providing the axis upon which the operation should be performed.

$${}^{i-1}_{i}T = R_X(\alpha_{i-1})\, D_X(a_{i-1})\, R_Z(\theta_i)\, D_Z(d_i)$$

This is a powerful concept which facilitates the analysis of any robot whose joints are based on revolute or prismatic members.

2) Forward & Inverse Kinematics: Forward Kinematics (FK) is the study determining the final pose of a manipulator given the individual joint positions and velocities (whether revolute or prismatic). By using DH parameter based transformations, the position or velocity of any joint can computed against the given frame of refernce (often the 0 frame). Forward kinematics is important since it provides a mapping from **joint space** to the **task space** of the robot.

Inverse Kinematics (IK) is the study of the joint positions given the position and velocity of the manipulator. IK is typically a harder task to perform than FK since there are often multiple solutions which satisfy the pose of the end effector.

Both forward an inverse kinematics are important for simulating joint positions and motion of a robot and determining which goal positions are feasible, respectively.

B. Path Planning vs Trajectory Generation

While frequently used as synonyms, **path planning** and **trajectory generation** (in the strict sense) describe two very quantities. A path is typically defined as a function of position connecting some point A to another point B. Paths are independent of time and therefore, cannot encode information such as velocity or acceleration, that depend on time. Trajectories, on the other hand, are paths with a dependency on time. Trajectory generation has substantially larger scope than path planning, and possesses substantial body of research.

While trajectory generation is typically of greater utility, as it provides more information about the behavior of system, for the context and scope of this paper, path planning will be a sufficient to describe the behavior of the system from a starting pose to an ending pose.

IV. Methodology

The methodology followed in this paper consists of deriving the governing equations for the robot, formulating the optimal control problem for MPC, designing the appropriate cost function for the desired behavior, translating this information into CasADi syntax, and numerically simulating the behavior of the robot under MPC control and using the outputs of the MPC to generate paths. Each of these elements is broken down in the following sections.

A. Formulation of Robot Dynamics

For puposes of this paper, the robot model only includes physical dynamics of the system and not the dynamics of the actuators. It would be relatively simple to derive the actuator dynamics for a more realistic model of how the system operates, but for the purposes of time and simplicity, it is assumed that the joint motors have perfect torque control.

As previously mentioned, the model of the robot excludes rigid body effects and only considers the effect of point masses rigidly connected with massless links. Under these assumptions, we can ignore contributions by moments of inertia. This simplifies the process of deriving equations of motion.

The Euler-Lagrange Equations are used to derive the explicit dynamics of the system. This variational approach is often more useful and scalable to large and coupled system than classical force and moment balances, since it this method only requires computing the kinetic and potential energies of the system and computing derivatives with respect to generalized coordinates. While equivalent to the Newton-Euler Equations, Lagrange Equations do not require the computing of accelerations. The general form of Lagrange Equations are.

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\boldsymbol{q}}_i} - \frac{\partial L}{\partial \boldsymbol{q}_i} = \left[\begin{array}{c} \tau_i \end{array}\right] \qquad \text{for i} = 1, 2, ... \tag{3}$$

Where the lagrangian L is the difference between the total kinetic and total potential energy of the system.

$$L = k - u \tag{4}$$

and where k and u are defined be to...

$$k = \sum_{i=1}^{n} k_i \tag{5}$$

$$u = \sum_{i=1}^{n} u_i \tag{6}$$

It should be noted that when defining the individual potential and kinetic terms, they must be written with resepect to the correct frame of reference $\{i\}$. This can be easily accomplished by using DH parameter homogeneous transformations, previously defined, to write the velocity or position vector of a particular link of the robot with respect to the correct reference frame.

$$k_i = \frac{1}{2} m_i v_{C_i}^T v_{C_i} + \frac{1}{2}^i \omega_i^T C_i I_i^i \omega_i \tag{7}$$

$$u_i = -m_i^0 g^{T0} P_{C_i} + u_{ref_i} \tag{8}$$

By computing i Lagrange equations, as expressed above, the equations of motion for the system will be determined. While simple, this process depends on the ability of taking derivatives, which for robotic systems can be highly coupled and result in massive expressions. Typically this would be done using a symbolic math libary or by using the itterative form of the Newton-Euler Equations instead.

In general, serially articulated robots with revolute joints will produce equations of motion (EOM) that take the following form.

$$\tau = M(\Theta)\ddot{\Theta} + V(\Theta, \dot{\Theta}) + G(\Theta) + F(\dot{\Theta}) \tag{9}$$

In this formulation, M is the mass matrix which models the effects of mass and moments of inertia which are related to the angular accelerations of the joints. The V vector models the centrifugal and Coriolis forces which are typically functions of velocities, while the G vector models the effects of gravity. Finally, the F vector is added to explictly include the effects of friction/damping, at the joints of each link.

$$M(\theta) = \left[\begin{array}{cc} m_{11} & m_{12} \\ m_{21} & m_{22} \end{array}\right] \tag{10}$$

Where

$$\begin{aligned} m_{11} &= m_1 L_1^2 + m_2\left(L_1^2 + 2L_1L_2\cos(\theta_1) + L_2^2\right) + \varepsilon \\ m_{12} &= m_2\left(L_1L_2\cos(\theta_2) + L_2^2\right) \\ m_{21} &= m_2\left(L_1L_2\cos(\theta_2) + L_2^2\right) \\ m_{22} &= m_2 L_2^2 + \varepsilon \end{aligned}$$

$$V(\theta,\dot{\theta})=\left[\begin{array}{c}-m_2L_1L_2\sin(\theta_2)\left(2\dot{\theta}_1\dot{\theta}_2+\dot{\theta}_2^2\right)\\ m_2L_1L_2\dot{\theta}_1^2\sin(\theta_2)\end{array}\right] \quad (11)$$

$$G(\theta)=\left[\begin{array}{c}(m_1+m_2)\,L_1g\cos(\theta_1)+m_2g_2\cos(\theta_1+\theta_2)\\ m_2gL_2\cos(\theta_1+\theta_2)\end{array}\right] \quad (12)$$

$$F(\dot{\theta})=\left[\begin{array}{c}c_f\cdot\dot{\theta}_1\\ c_f\cdot\dot{\theta}_2\end{array}\right] \quad (13)$$

It should be noted for simulation purposes, the equations of motion need to express τ as the input to the robot and the angular accelerations $\ddot{\Theta}$ as the outputs.

$$\ddot{\theta}=(M^{-1})[\tau-V-G-F] \quad (14)$$

However, expressing the governing equations in this form result in numerical instabilties during simulation which arise from inverting the mass matrix $M(\Theta)$. Since the mass matrix is a function of angular positions Θ, specific configurations can cause the mass matrix to near a singularity, when inverted. If evaluated numerically, the computed value will explode, typically resulting in the value becoming **INF** or **NAN**. To avoid this, the value of ε can be added to the diagonal terms of M to garuntee that the system will never become singular, during inversion.

By evaluting [eq.14], it is possible to write the nonlinear state space for the robot, as a function of the states $x_1=\theta_1$, $x_2=\theta_2$, $x_3=\dot{\theta}_1$, and $x_4=\dot{\theta}_2$.

$$\begin{aligned}\dot{x}_1&=x_3\\ \dot{x}_2&=x_4\\ \dot{x}_3&=\ddot{\theta}_1\\ \dot{x}_4&=\ddot{\theta}_2\end{aligned} \quad (15)$$

Where $\ddot{\theta}_1$ and $\ddot{\theta}_2$ are defined in APPENDIX???????????.

While still highly nonlinear, [eq. 15] encapsulates the complete dynamics of the robot. This formulation can be directly used for numerical simulatation and for use in developing the constraints neccessary for model predictive controller.

B. MPC Formulation

The mathematical basis for MPC stems from the standard optimal control problem. Under the assumptions that an internal model of the system dynamics exist, that the prediction horizon is finite and proceeds in a receding fashion, the inputs applied to the system, by the controller, are piecewise constant at each time-step, and where the value of the current input is obtained by taking the first value of the optimal input sequence which is obtained by solving the optimization problem at each time-step. This can be written in standard form as...

$$\begin{aligned}&\min_{n} J_x(\mathbf{x}_0,\mathbf{u})=\sum_{k=0}^{N-1}\ell(\mathbf{x}(k),\mathbf{u}(k))+\ell(\mathbf{x}(N),\mathbf{u}(N))\\ \text{s.t: }&:\mathbf{x_n}(k+1)=\mathbf{f}(\mathbf{x_n}(k),\mathbf{u}(k))\\ &\mathbf{x_u}(0)=\mathbf{x}_0,\\ &\mathbf{u}(k)\in U,\forall k\in[0,N-1]\\ &\mathbf{x_u}(k)\in X,\forall k\in[0,N]\end{aligned} \quad (16)$$

In this formulation, the cost function is only propagated to the end of the prediction horizon N, from the current initial state of the system. The first equality constraint restricts future states of the system to be feasible with respect to the system model. The second equality constraints requires the initial state of the solver to be the current initial state of the system, while the final constraints require the inputs and states determined by the solver to be valid elements of feasible input and state sets respectively.

It should be further noted that this paper uses the **Multiple Shooting** implementation of MPC as this approach has shown to produce better results and obtained solutions far faster than the naive single shooting approach. This means that the optimization problem has been lifted into a higher dimensional space by including the each state and input over the prediction horizion as a decision variable for the solver.

1) Cost Function Design: The general stage cost, as each time-step of the prediciton horizon utilized in this paper follows the typical quadratic form.

$$\ell(\mathbf{x_v}(k),\mathbf{u}(k))=(x-x_d)Q(x-x_d)+(u-u_d)R(u-u_d) \quad (17)$$

This parameterization of the stage cost can be further simplified using norm notation.

$$\ell(\mathbf{x_v}(k),\mathbf{u}(k))=\|x-x_d\|_Q^2+\|u-u_d\|_R^2 \quad (18)$$

The quadratic nature of this cost function is eligant in how simple and intuitive it is comprehend as as well how simple it is to tune using the matrices Q and R.

However for the scenario testing **obstacle avoidance**, it is neccessary to append the origin stage cost with an addition term. By adding an obstacle penalty to the stage cost the controller is incentivized to avoid letting the end effector of the robot get too close the specified object. By specifying circular envelopes around the end effector and the obstacle, it is possible to define the minimum distance between the two objects such that they do not approach any closer to each other than the sum of radii which define the circular boundaries. This can be expressed mathematically by computing the euclidean distance between the obstacle and the end effector.

$$\text{min. dist}=-\sqrt{(x-x_0)^2+(y-y_o)^2}+(r+r_o)\leq 0 \quad (19)$$

Where x and y constitute the current 2D position of the end effector, while x_0 and y_0 constitute the stationary position of the obstacle. The values of r and r_0 define the radii of the circular envelopes enclosing the end effector and obstacle respectively.

By using this measure, it is possible to create a penalty such that if the

$$\mathrm{C}(P, P_0) = S(max(\text{min. dist}(P, P_0), 0))^2 \tag{20}$$

Where S is a large positive weight, and P and P_0 are the points defining the end effector and obstacle positions respetively. If the inequality is every violated and the minimum distance between the obstacle and end effector is greater than zero, the magnitude of the distance violation will be used as a penalty in the cost function effectively incentivizing the controller to avoid letting the controller and the obstacle become too close to each other. If the inequality holds, then the value of zero will choosen in the max() function and will not contribute to the cost function. The final stage cost for obstacle avoidance results in.

$$\ell(\mathbf{x_v}(k), \mathbf{u}(k)) = \|x - x_d\|_Q^2 + \|u - u_d\|_R^2 + C(P, P_0) \tag{21}$$

While crude, this formulation enables the rudimentary form of obstacle avoidance, if the position of the obstacle is known by the controller. One of the conns of using this penalty in the control law is that the controller is not strictly speaking forbidden (constrained) from violating this minimum distance, but rather is only incentivized by a large penalty to not violate the condition. This is a subtle difference, but time restrictions prevented further exploration of more rigorous constraint formulations such as the use of slack variables. [6]

C. Controller Development in CasADi

In order to solve the optimal control problem, defined above, the CasADi library was utilized to simplify the process of solving optimization problems by use of its symbolic functionality as well as its automatic differentiation, which allow a simple and direct casting of the mathematics into a well structured programming syntax.

By leveraging this library and its functionality, it becomes a simple and direct task to cast the MPC formulation into meaning code which can then be feed into a standard optimization solver. For the purposes of this paper, the **IPOPT** (Interior Point Optimizer) solver was used since it comes prepackaged with CasADi and performs well in a variety of applications.

D. Controller & Planner Simulation

Once MPC is implemented via CasADi, numerical simulation merely requires using the first optimal input of the solver, at the initial state, applying that input to a discretized system model, computing the resulting states, updating the current state of the controller, and reapplying the solver for the duration of the simulation time.

By virtue of using muliple shooting, the decision variable for this MPC formulation not only includes the inputs over the prediction horizon, but also the corresponding optimal states, according to the model. For the purposes of planning (in simulation) it is possible to directly copy the optimal predicted states directly from the solvers solution, at each time-step, and to use these state sequences as the optimal path, over the duration of the simulation.

E. Controller Testing

Since the first objective of this paper is to test the efficacy of MPC, there are several interesting scenarios which are of general interest when evaluating how effective MPC based control is, on this type of system.

1) Pose-to-Pose Control: The primary task of any controller is to drive the system from its initial condition to a desirable end result. For MPC on a robotic arm, this is equivalent to determining whether MPC can guide the robot from any initial pose (position and orientation) to any other arbitrary feasible pose without driving the system in to actuator saturation, or causing the system to violate any constraints or performance objectives. To test this in simulation, MPC will be used to control both of the actuated links independently of each other, to achieved the desired final pose.

2) Model/Parameter Mismatch: An important characteristic of any controller is its robustness to uncertainty. Since MPC makes heavy use of a system model for prediction, one important test is to determine how MPC performs when the system being controlled and the model used in the controller a mismatched. To test this in simulation, the system and controller models are intentionally mismatched, such that the system states are continually used to update the controller.

3) Obstacle Avoidance: In robotics and autonomous systems, it is often desireable to design a system that is capable of intelligently using sensory data to predicted the best way to acheive its objective, without external intervention. For robotic manipulators, a common task is for the end effector to avoid collisions with obstacle in its workspace. This paper will test the ability of MPC to avoid obstacle by tailoring the cost function to maintain specified working distance way from a static object in its path.

F. Planner Testing

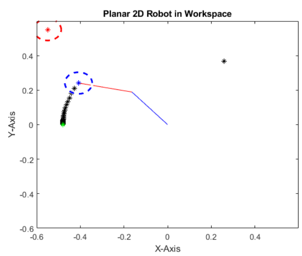
The second primary objective of this paper is to implement MPC such that the state predictions can be used to plan the path which the end effector takes, from one pose to another. While this paper will not go into depth about the application of path planning for robotic manipulators, it intends to cover how to obtain a continuous path from discrete state predictions supplied by MPC for simple motion from one feasible pose to another, as well the simple obstacle avoidance metioned above.

V. Results

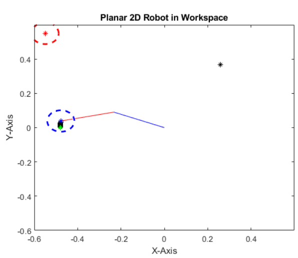
A. Controller Results

1) Pose-to-Pose Control: As anticipated, MPC performed very successfully in the simplest and most direct usecase for the RR-robot. Under nominal conditions, with a perfect system model, a clear workspace, and few extraneous system constraints, MPC was consistently able to drive the robot from one pose to another. The behavior seen in [Fig.3] is representitive of the typical behavior of this system. Under these slightly idealized conditions, the controller was able to locate the end effector of the robot within a tolerance of smaller than .001% of the desired reference state.

(a) Pose at .5 seconds



(b) Pose at 1.5 seconds



(c) Pose at 3 seconds

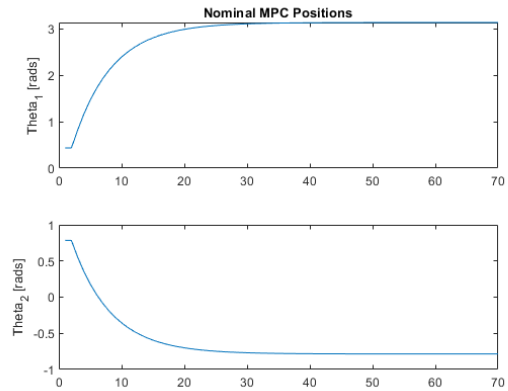
(d) Pose at 6 seconds

Fig. 3: Pose-to-Pose control via MPC

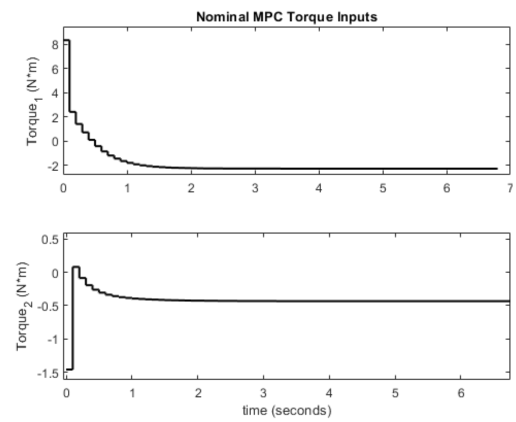
On average, over a random sampling of initial and final manipulator configurations, the MPC solver took between 8-10ms to compute the control action for the next time-step. This roughly equates to a nominal 80-120Hz control frequency. Given that time-step used for simulation is 50ms, this MPC controller is more than sufficient for presumed operating frequency of the system. The plots in [Fig.4] show the evolution of the system under the action of the controller, for the simulation depicted in [Fig.3].

It should be noted that the torques of the system never converge to zero, even though that would be a desirable. This behavior is due to the fact this RR-robot is standing in the vertical plane and is subject to gravity. This requires the controller to maintain a bias torque at each of the joints of the robot, to compensate for the effect of gravity. If this same robot was oriented oriented parallel to the horizontal plane, these bias torques would not be neccessary.

Additionally, due to gravity, certain configurations of initial and goal positions will cause cause the actuator effort to increase or decrease due to the force of gravity working for or against the controller.



(a) Angular Position vs Time



(b) Torque vs Time

Fig. 4: Pose-to-Pose MPC Curves:

2) Model/Parameter Mismatch: Another primary result obtained is the impact which Model/Parameter Mismatch has the steady state performance of the controller. During simulation, two seperate sets of masses were parameterized for the MPC model and the simulated system. Similar to the previous result, MPC was setup to perform pose-to-pose control of the robot, with the only difference being the masses used in the controller and in the system. The following table details the percent steady state error witnessed using the standard MPC formulation, including a terminal cost.

TABLE I: Plant Model Mismatch

	Mass 1 (kg)	Mass 2 (kg)	% Error θ_1	% Error θ_2
Model Params.	.5	.25		
System Params.				
(+5%)	.525	.2625	1.14%	-.59%
(+10%)	.55	.275	2.28%	-1.20%
(+15%)	.575	.2875	3.43%	-1.93%
(+20%)	.6	.3	4.58%	-2.74%
(+25%)	.625	.3125	5.74%	-3.63%
(+30%)	.65	.325	6.88%	-4.59%

For the sake of brevity, only the cases were both masses are changed by the same percentage are included in [Table I]. This treatment roughly approximates a constant measurement or estimation bias when specifying parameter values. This bias is varied beteen 5-30% to investigate how the system performs, even when the model is vastly different from the actual parameters of the physical system.

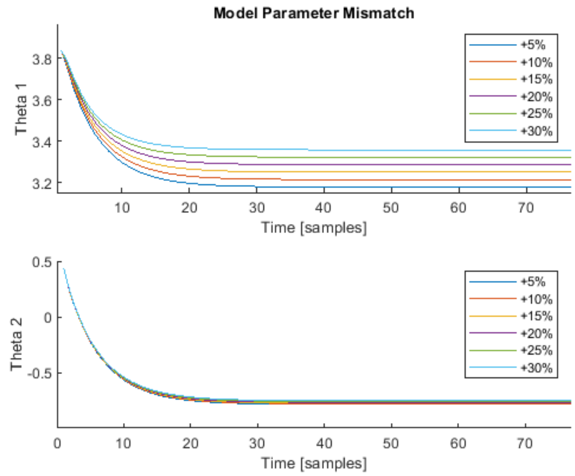


Fig. 5: Model Parameter Mismatch: Steady State Solutions

From the results in [Fig.5], all else being equal, the mismatch between the system and controller models has the potential to effect the steady state performance of the robot. In fact, [Table.I] demonstrates that the magnitude of the parameterization mismatch is proportional to the magnitude of the steady state error.

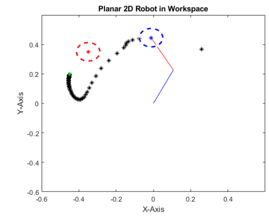
3) Obstacle Avoidance: Probably the most interesting application and results obtained from this paper, concern the use of MPC for obstacle avoidance. It was slightly disappointing to see that, as a whole, the performance of obstacle avoidance greatly varied greatly throughout simulated testing.

While the controller was successfully able to avoid objects which were directly in the most convient path between the initial state and the goal, many of these trials resulted in oscillatory behavior of robot, theoretical collisions between the end effector and the obstacle, and/or the robot never reaching its desire final pose. Even with static objects, the success of the controller oftened appeared predicated on the relative positions of the end effector and the nearest object.

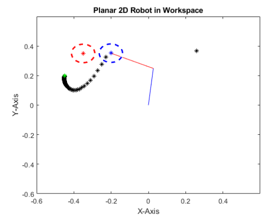
In [Fig.6], the robot arm can be seen circumventing the red static object, in its path. However, it should be noted that the initial state prediction horizon (seen as black *) violates the collision penalty, and that it is only as the robot nears the obstacle that MPC reroutes the predicted path. This behavior corresponds to the increasing penalty which the cost function enforces as the distance between the robot and the object gets smaller. Due to this behavior, it is believed that the formulation of obstacle avoidance as part of the cost function and not an explicit constraint on the optimization problem, which MPC is runnings is at least part of the reason that obstacle avoidance was finicky and severly inconsistent.

One side effect of this delayed action to prevent collision is that, typically, this sudden rerouting is often accompanied

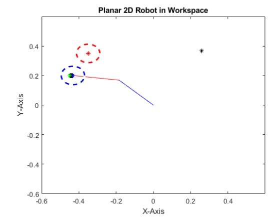
(a) Initial Pose



(b) Avoid Obstacle



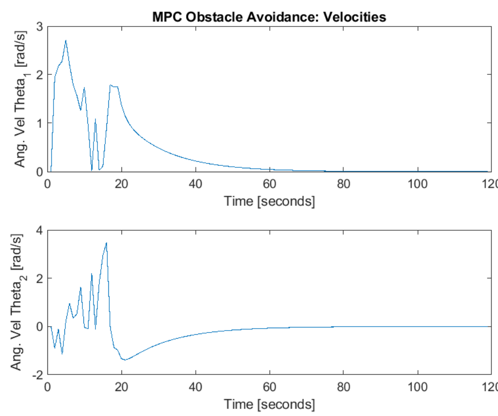
(c) Avoid Obstacle



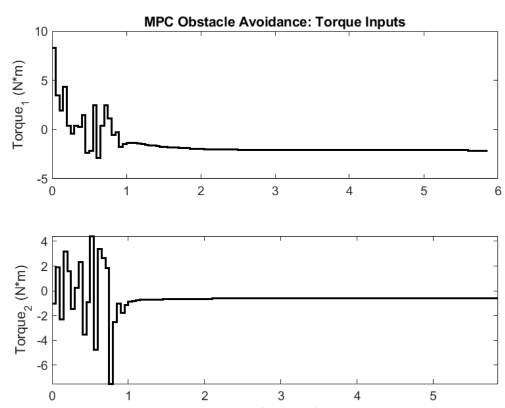
(d) Final Pose

Fig. 6: Obstacle Avoidance Control System

by large commanded torques. While not an explicit problem, these large torques often result in significant velocity pikes. Depending on the application of the robot, these sudden movements may not be undesirable.



(a) MPC Obs. Avoidance: Angular Vel.

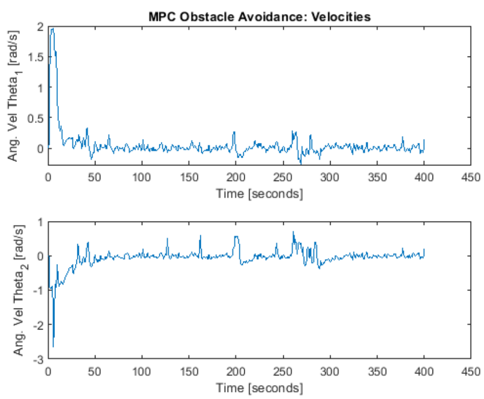


(b) MPC Obs. Avoidance: Actuator Torques

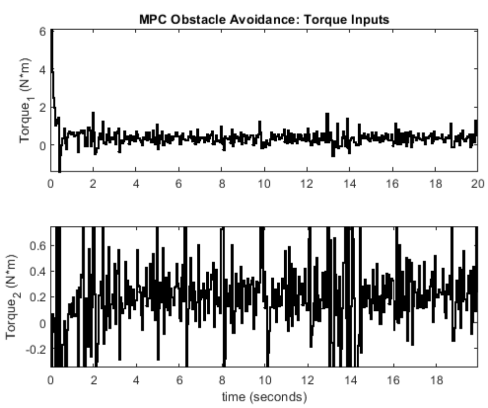
Fig. 7: Obstacle Avoidance Control via MPC

Notice in [Fig. 7], the sudden changes in velocity, which roughly correlate to the large swings of the actuator action, on the robot.

As shown in [fig.6], when obstacle avoidance (under its current cost function formulation) does work, it is very compelling. By the simple addition of an extra term in the objective function or an added constraint, the controller can achieve nontrivial goals which other control strategies would have required explicit planning to reproduce. In this way, MPC with obstacle avoidance presents a very neatly packaged general methodology for hitting control objectives without designing custom planning to avoid undesirable states.



(a) Obs. Avoidance: Oscillating Velocity



(b) Obs. Avoidance: Oscillating Torque

Fig. 8: Nonresolving Obs. Avoidance Oscillations

However, as previously mentioned, the results sean under the current formulation for obstacle avoidance via MPC are mixed at best. As seen in [fig. 8], the system is often unstable and does not resolve to any steady state value, but merely oscillates, while the controller is stuck between getting to the final pose and avoiding the obstacle with the smallest possible deviation from the original optimal path. Even though other solutions exist, the controller appears stuck in a local minimima. Even if the prediction horizon and MPC gains are tuned, this behavior appears to persist or the controller is so tuned that the penalty for collision is not becomes insignificant and the resulting control action slams the obstacle in on its way to the goal pose. This problem would benefit from a more rigorous and sophisticated treat of MPC based control with obstacle avoidance

B. Planner Results

The basic premise behind path planning using MPC is to utilize the optimal states over the prediction horizon as a guide to define a continuous path. In this paper, this is accomplished by using the extracting the optimal states, from within in the controller, which are then used to define cubic splines from the inital pose to the final pose. This technique can be applied to either the joint space or the end effector space of the robot, providing several ways in which this path can be utilized.

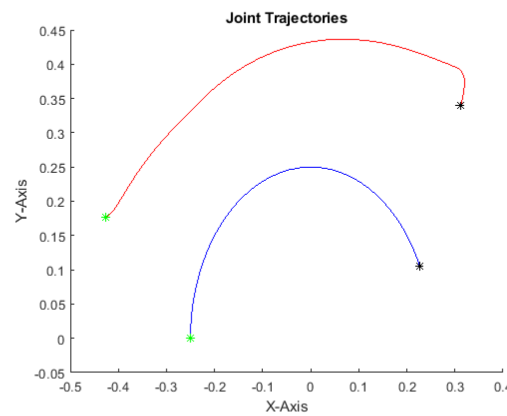


Fig. 9: Joint Paths

1) Joint Paths:

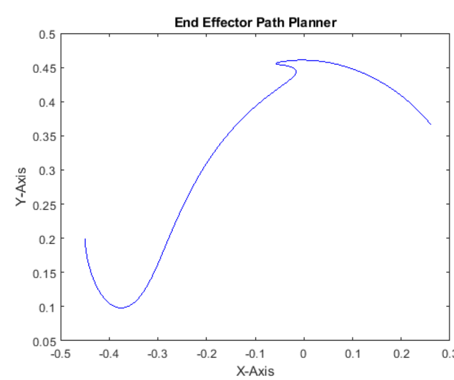


Fig. 10: Joint Paths

2) End Effector Paths :

VI. Discussion

A. Control

After testing the system, it is clear that MPC is a very straightforward control scheme to apply to robotic systems in free space. With the ability to work on large MIMO systems (e.g. robotic manipulators) is a very intuitive and practical benefit for controller design and implementation. In particular, since most robotic manipulators are sensored and can provide highly accuracy state measurement, without state estimation, MPC is definitely an attractive control strategy when extraneous constraints or limitations are not placed on the system.

However, from the testing performed in this paper, it does appear that the addition of obstacles does significantly complicate the easy of using MPC for control. While there are many configurations of the end effector and the obstacle, that work seamlessly, other required significantly more computation and often failed to guide the end effector around the object, often resulting in very oscillatory behavior or inducing rapid wipping motions of the manipulator, to bypass an object.

This is behavior is likely a result of a poorly design cost functionl. As mentioned previously, the trade-off between constraining the system for colliding with the object versus heavy penalizing the controller if it collides with the object is unclear to the author, and would require deeper research. However, from the simulation data collected, it appears that by implementing a obstacle collision penalty in the stage cost, that the controller **could** be presented with situations where the penalty incurred is insufficient to prevention collision.

Another aspect about using MPC for obstacle avoidance is the difficulty in tuning the controller such that it could be effectively utilized on a wide number of configurations. This is most notably seen in conditions where the controller actually avoids the desired obstacle, but proceeds to oscillate due to poor tuning.

The last feature of MPC that was covered in this paper is a mismatch between the parameterization used by the controller and by the physical system. As seen in this paper, even with a terminal constraint, when the masses of the robot arm are different from the model used for MPC, the controller does not converge to the desired pose, and exhibits a steady state error, proportional to the magnitude of the mismatch.

This is most likely the result of the estimated reference input. This torque reference is seen in the last term of [eq.18] by the term u_d. The reason for this is that the modeled system is used to predict the steady state torques required at the final state, by using [eq.9]. Since this operation uses the system model, it must also be mismatched, which corresponds to a steady state bias. While this effect can be minimized by tuning the MPC gains, this could produce other undesirable effects like actuator saturation.

B. *Planning*

Possible To use MPC as a supervisory controller/planner for other

MPC planner is able to update the best path (given the current state) and can account for model uncertainties or object/obstacles in the direct path.

Planning with MPC allows the planner to track smoothly though the optimal state horizon which can then be leveraged by other controllers ...etc. which function at faster speeds than the base MPC controller could.

Unlike the most commonly used path planning algorithms (A*, Djikstra, RRT, RRT*, PRM, D*, and other discretized, graph, or heuristic search algorithms, such as Artifial potential fields) Path planning with MPC is a very control theoretic approach.

While the use case seen in this paper is not very impressive, predicting and controlling the path which the end effector of a robot takes from one position to another can quicker be seen in the three dimensional case where discretizing the entire feasible can be computationally intensive.

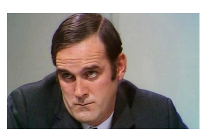
VII. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank Mountain Dew and his matress for constant support and comfort.

[4] [7] [10] [1] [9] [8] [5] [2] [3] [11]



Jonathan Dorsey I've already told you once. It is an ex parrot. Its has ceased to be. Recieved his Bachelors Degree in Mechanical Engineering from the San Jose State University. With a focus on mechatronics and control systems, he has developed an interest in reinforcement learning, computer vision, and control and design of autonomous systems.