School of Engineering, Discipline of Mechanical Engineering and Mechatronics, pp. 1–58

Beam type 3D Printer

MECH4841 Final Year Project Report

December 2020

Abid Khan [1]

[1] *Student of Mechatronics Engineering,*
The University of Newcastle, Callaghan, NSW 2308, AUSTRALIA
Student Number: 3189548
E-mail: c3189548@uon.edu.au

Abstract

This report covers the design aspect of a tower crane-based 3D printer. It includes the energy based mathematical modelling the dynamics of a fully actuated system using Euler-Lagrange equations of motions. The system consists of one rotational and three prismatic joints. It also covers the forward in inverse kinematics for the project along with two control designs which is linear quadratic regulator and model predictive control. It also covers a way for generating a trajectory with constraints and slew rate limits. This report will discuss the results produced by the simulations of the two control designs.
The future work for this project would be the implementation all the control actions on the hardware and compared with the traditional gantry-based 3D printer hardware.

Dot Point Summary

- Applied euler lagrangian modeling methods to simulate a beam type 3D printer.
- Applied forward and inverse kinematics.
- Applied trajectory generation techniques to create a trajectory from waypoints
- Applied linear quadratic regulator on the model in simulation.
- Applied non-linear model predictive controller on the model in simulation.
- Comparison between the two controllers designed.

Acknowledgements

Dr Alejandro Donaire has provided significant amount of help and support with this project. He is also the supervisor for this project. I'd also like to thank my family and friends for providing the support and stability I needed for the successful completion of this report. Some of those people are listed below.

- Dr Bill McBride for his faith.
- Patrick Harper for his help.
- Muhammad Hasham for his help.
- Paul Hughes for his help.

Contents

List of Figures

List of Tables

Chapter 1

1. Introduction

The project aims at design and simulation of a 3D printer for house construction. The project is focused on the modelling, analysis, simulation and control and evaluation of a rotational machine with prismatic arm, the 3D printer, that moves in its workspace where the walls of the house are to be printed. An active counterweight balancing system is attached to the printer to increase the precision, stability and speed of the machine. This report will also work on comparing different controllers to find the appropriate solution for the project.

2. Problem Formulation

2.1. Use of 3D printing for houses

Previous design – full frame design

Traditionally, 3D printers use a gantry-based design which operate by following a precomputed path and extruding a layer of material upon a work-surface for multiple layers to create a desired object. This filament is extruded out through a nozzle, whose movement is precisely controlled by a computer which operates motors that slide the nozzle along the X gantry, the X gantry in Y gantry and both gantries along the Z gantry.

Figure 1: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

This framework has been applied by Fused Deposition Modelling (FDM) 3D printers and has shown promising results.
However, this method has few limitations:

- Size of the gantry system required increases dramatically when considering the construction of larger buildings (it must be larger than the building to fully encompass it).
- Construction industry is a very conservative market that is slow to change. It would be easier to use a system that is tried and tested in a construction environment.
- Due to the bulky structure require it to be disassembled For transportation purposes.

Therefore, a methodology to use tower cranes to perform 3-D printing was proposed.

Use of tower crane design

The tower cranes have been in use for over a century. [**12**] It has proven to be a versatile method for transporting and handling heavy materials across the building and its working space. This has led to an idea that if it could be utilised for 3D printing houses since its robust nature and ease of setup and transportation as compared to the traditional gantry type 3D printers. The 3D Printer shown in figure 1 below provides an insight into the 3D printing utilising and old and robust concept.

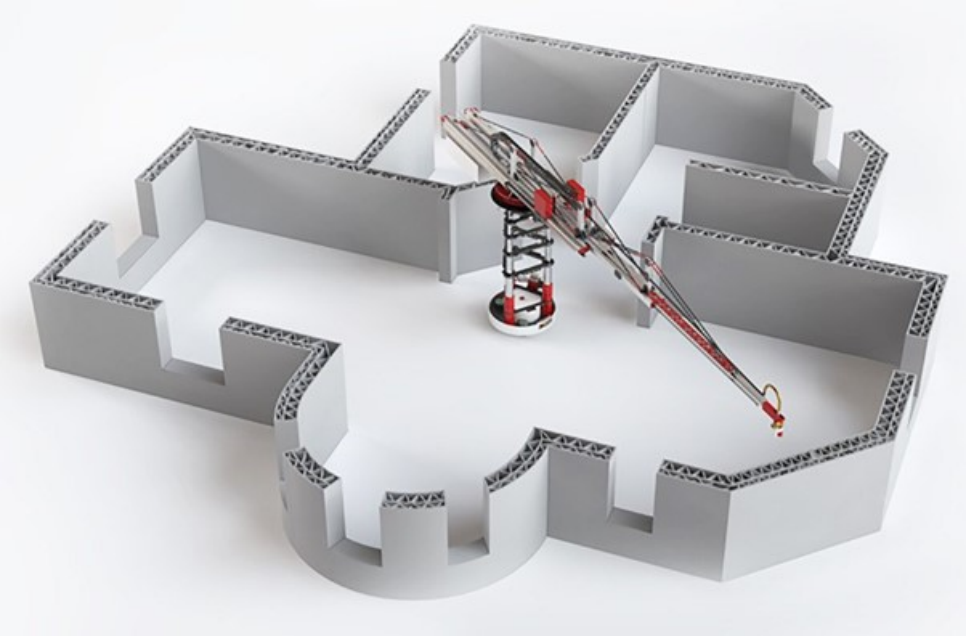

Figure 2: Apis Cor's 3D Printer shown printing walls for a house retrieved from [**13**]

Although some 3D printers have already been designed, built and have demonstrated proved results, they lack a counterbalancing system for the stability of the machine. Without such a stability system, the system must be constrained from achieving higher speeds and the support to be made stronger and heavier to keep the machine stable.
This problem can be solved by creating a counterbalancing system that could further stabilise the structure by keeping the center of gravity close to the beam's centre. This will also help provide better precision as the system moves, thus resulting in better repeatability.

2.2. In-scope

- Modelling Tower Crane.
- Controlling Tower Crane.

2.3. Out-of-scope

- Proof of concept – Friction and damping were not considered.
- Assumed perfect motors with output and slew rate limits .

- Assumed perfect encoders and thus perfect state estimation – no observer or state estimation considered.

2.4. Crane Systems

What does a crane look like? (insert picture)
How do cranes work? (counterbalance ect)
Use of cranes in construction
Humans get tired
Mistakes happen
Humans require food
Humans cant get whipped without quitting
Robots cant unionise

2.5. Safety Considerations

Can you prevent over-swinging? Oscollations in control are dangerous for cranes, does this mean we need smart control algorythms Moments balancing

3. Additive Manufacturing

Additive manufacturing (AM), also known as Additive Layer Manufacturing (ALM), is commonly used in industries for 3D printing. In general, it is a process controlled by the computer by depositing the material usually in layers. A number of various AM processes include Binding Jetting, Direct Energy Deposition, Material Extrusion, Powder Bed Fusion, Sheet Lamination, Vat Polymerization and Wire Arc Additive Manufacturing.
Hideo Kodama of Nagoya Municipal Industrial Research Institute was the earliest to invent 3D printing manufacturing equipment; he invented two additive methods for fabricating 3D models. Many materials are currently used such as biochemicals, ceramics metals, thermoplastics and specialised concrete. AM is used across a growing number of industries creating a wide range of products; some of these industries include aerospace, automotive, construction and medical.
AM was initially considered to be only used for rapid prototyping however the increase in precision and repeatability has led it to be used in everyday manufacturing This is due to cheaper material availability and more generous support and machines availability in the market. There are significant advantages that can be achieved from this process which includes printing hollow and intricate structures.
This project aims to design an AM machine that could print houses, which would lower the environmental footprint by reducing the material used for construction while also providing cheaper housing solutions. By minimising the use of the material as these structures can be printed hollow, it will also benefit people living in areas where wood and other raw materials are scarce. Since this is a relatively new process, there are ways it can be improved. One issue is the transportation of the machine as the structure could be bulky and hard to assemble.
This issue could be resolved using a beam type 3D printer and a counterbalancing mechanism for increased stability. Unlike traditional construction techniques, complex structures could be made as

the machines can achieve better repeatability and precision than humans.

With the advancements in composite materials like concrete, special additives can be added to concrete to set it in mere minutes which used to be days. This will advance the construction process to be completed in a matter of days rather than a few months. This will reduce the cost of the construction significantly and deliver the houses faster, in order to achieve this, a control system for a beam type 3D printer is required to be made.

4. Automation for Precision manufacturing

In the modern days, manufacturers are pushing for tighter tolerances faster turnarounds and cheaper components. This has led to the use of advanced automation in widespread applications in an increasing number of industries. This not only increases parts repeatability in micro or ultra-precision machining processes, but this also significantly reduces the labour required to produce a part this in turn results in a per piece cost reduction, especially in higher volume applications. This also insulates the customer against future price volatility due to the labour market. Some standard implementations of control theory in this context are Automated lathes, CNC machines and 3D printing devices.
It is control theory that allows controlling of dynamical systems in various engineered processes using machines. The objective is to make a control system that enables us to precisely coordinate control actions that optimise tool pathing, reduce delay, overshoot, and ensure control stability. The controller that works by minimising the difference between the desired value and the actual value is called error which is applied as feedback to initiate a control action which works intuitively to reduce the value between desired and actual value.
Control theory allows us to achieve repeatability even when faced with a non-linear or highly complex system.

5. Dynamic Systems

Before proceeding, it is important to know what dynamic systems are. A system whose current state produces its following state by a certain principle of change and thus produces a trajectory in state space is called a dynamic system. Consider a system where the derivative of the states is a function of the states only:

$$\dot{x} = f(x) \tag{5.1}$$

Equation 5.1 represents an autonomous dynamic system as the next state is a function of the current state. Successive application of this expression generates a sequence: From **(Khalil, 2002)**

$$\dot{x} = f\left(x\right) \rightarrow \ddot{x} = f\left(\dot{x}\right) \rightarrow \dddot{x} = f\left(\ddot{x}\right) \rightarrow \ldots \tag{5.2}$$

This sequence can be as long or short as one wishes but an important form of change in the system is the zero change where the rate of change in the states becomes zero i.e., stability, where the dynamics

actively recreate the preceding state into current state. This means that for a certain equilibrium point $x_c = 0$, the function $f(x_c) = 0$. The equilibrium point can be stable or unstable, the criterion for is explained below. From **(Khalil, 2002)**

The equilibrium point of the system is:

- Stable if, for each $\epsilon > 0$ there is a $\delta = \delta(\epsilon) > 0$ such that,

$$||x\,(0)|| < \delta \rightarrow x(t) < \epsilon, \forall t > 0$$

- Unstable if it is not stable.
- Asymptotically stable if it is stable and δ can be chosen such that,

$$||x\,(0)|| < \delta => \lim_{t \to \infty} x(t) = 0$$

6. State Space Models

A model that represents a physical system using its inputs, outputs and states as variables related to each other by a set of differential equations is called a state space model. The states of the system can evolve with time depending on their current values or the values of inputs introduced into the system over time. The states of the system can be represented as vectors in a confined space. The state space model is dependent on the type of system that we are dealing with. Generally, all physical systems are characterized into linear or non-linear systems, both of which are explained briefly in the next section.

7. Linear and Non-Linear Systems

The most general form of a dynamic system of n state variables is usually defined by an equation expressing the time derivative of the state variables as a function of time, t, the state variables, $x_1, x_2, \ldots, x_n$ and input variables $u_1, u_2, \ldots, u_m$. With $x = [x_1\ x_2\ \ldots\ x_n]^{\mathrm{T}}$, $u = [u_1\ u_2\ \ldots\ u_m]^{\mathrm{T}}$, it may be written.

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{t}, \mathbf{x}, \mathbf{u}) \tag{7.1}$$

Many systems do not have input variables, or they may be written as functions of the time and state variables. One class of such functions are linear systems which have the form:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} \tag{7.2}$$

where $A = [a_{ij}(t)]$ is an $n \times n$ matrix of time dependent functions. Solutions to such a system have the property that any linear combination of solutions is itself a solution. They form a vector space, the state space. The constant vector function $x_0 = 0$ is always a member of the state space. It has the property that $\dot{x}_0 = 0$ and is an equilibrium state of the system.

When each of the a_{ij} are constant, the general solution to the system usually has the form:

$$\dot{\mathbf{x}} = \mathbf{Be} \tag{7.3}$$

Here **e** is a vector, $[e_1(\mathrm{t})\ e_2(\mathrm{t})\ \ldots\ e_n(\mathrm{t})]^{\mathrm{T}}$, of time dependent functions where $e_i\,(t) = e^{\lambda_i t}$ and **B** an $n \times n$ matrix of constants. The λ_i are the eigen values of **A** which may be complex. The real part gives rise to an exponential function, the complex part is associated with a sinusoidal function. Any state is a linear combination of some, not necessarily all, of the functions in **e**. The behaviour of a state over time depends on the sign of the real parts of the eigenvalues contributing to the state:

- If any sign is positive, the size of the state grows exponentially. This is an unstable state.
- If no sign is positive but at least one associated eigenvalue has real part 0, the associated function in **e** is purely sinusoidal or constant. If the state starts close to the equilibrium state, $\mathbf{x_0}$, it will stay close to equilibrium over time. The state is stable.
- If all signs are negative, the state will be dominated by exponential decay and the state will tend towards the equilibrium state, $\mathbf{x_0}$. The state is asymptotically stable.

Most realistic dynamic systems are far more complex than linear systems entailing algebraic combinations of power, sinusoidal, exponential, and other nasty functions of the time and state variables. Analytic solutions are usually impossible and numerical methods must be employed. But the notions of stability carry over to these problems and it is often possible to decide when they give rise to stable and asymptotically stable solutions.

8. Energy Based Modelling

The dynamic equations of any mechanical system are obtainable from the renowned Newtonian classical mechanics. The disadvantage of this formalism is the usage of the variables in vector form, complicating substantially the analysis when increasing the joints or if there are rotations present in the system. In these situations, it is favourable to utilize the Lagrange equations, which have a formalism of scale, simplifying the analysis for any mechanical system. To use Lagrange equations, it is essential to follow four steps listed below:

1. Calculation of all kinetic energies.
2. Calculation of all potential energies.
3. Calculation of the Lagrangian.
4. Solve the equations.

Where the kinetic energy can be both translational and rotational, this form of energy can be a function of both, position, and the velocity $K(q(t), q(t))$.
The potential energy is due to conservative forces such as the forces exerted by gravity and springs, this energy is in terms of the position $U(q(t))$.
Is defined the Lagrangian as

$$\mathbf{L} \ = \ \mathbf{K} - U \tag{8.1}$$

Therefore, the Lagrangian is defined of the following in general terms.

$$\mathbf{L}\left(\dot{\mathbf{q}}, \mathbf{q}\right) = \mathbf{K}^{*}\left(\mathbf{q}, \dot{\mathbf{q}}\right) - \mathbf{U}\left(\mathbf{q}\right) \tag{8.2}$$

Finally, the Euler-Lagrange equations of motion for a system of i degree of freedom are defined as follows.

$$\frac{d}{dt}\left(\frac{\delta L(q,\dot{q})}{\delta q_i}\right) - \frac{\delta L(q,\dot{q})}{\delta q_i} = \tau_i \tag{8.3}$$

Where $i = 1, \ldots, n, \tau_i$ are the forces or pairs exerted externally in each joint, besides nonconservative forces such as friction, resistance to movement of an object within a fluid and generally those that depend on time or speed. It will be obtained an equal number of dynamic equations and DOF. **(Duarte Madrid, Henao and Querubín, 2017)**

9. Kinematics

A precise definition of kinematics is found on Oxford dictionary which states,

> "Kinematics is a branch of mechanics concerned with the motion of objects without reference to the forces which cause the motion". **Kinematics. (2020). In Oxford Online Dictionary**

9.1. Forward Kinematics

Forward kinematics is a process of calculating end effector's position and velocity by using known joint angles and displacements. While it is mostly used for an end effector position it could also be used for any frame in space with respect to a reference.

Denavit Hartenberg

As the name prescribes, Denavit Hartenberg convention was introduced by Jacques Denavit and Richard Hartenberg in 1955.The purpose of this convention was to standardise the coordinate frames for spatial linkages. The term Denavit Hartenberg is widely known in the engineering fields.

There are four parameters associated with this convention. This technique is used for attaching reference frames to the links of a robotic manipulator or kinematic chains.

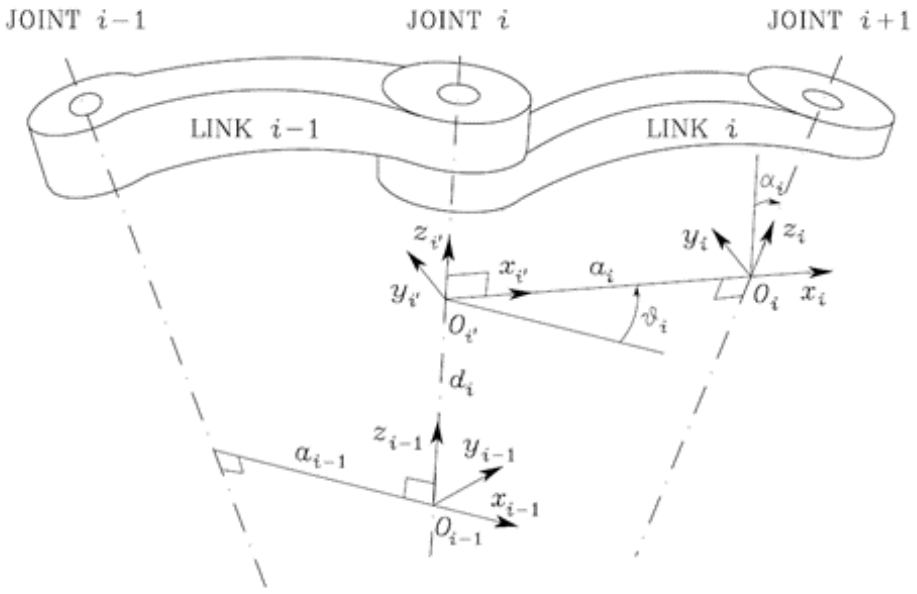

Figure 3: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

With those four parameters, coordinates can be translated from $O_{i-1}X_{i-1}Y_{i-1}Z_{i-1}$ to $O_iX_iY_iZ_i$.

d: offset along previous z to the common normal
θ: angle about previous z, from old x to new x
r: length of the common normal (aka a, but if using this notation, do not confuse with α). Assuming a revolute joint, this is the radius about previous z.
α: angle about common normal, from old z axis to new z axis

$$A_n^{n-1} = \begin{bmatrix} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\cos\alpha_n & r_n\cos\alpha_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\cos\alpha_n & r_n\sin\alpha_n \\ 0 & \sin\alpha n & \cos\alpha n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & R & & T \\ & & & \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9.1}$$

By using the matrix above a homogenous transformation matrix is produced where R is a 3x3 rotation submatrix and T is a 3x1 translational vector.

9.2. Inverse Kinematics

Inverse kinematics is a process for determining the joint variable's value given an end effector's position and orientation. The problem of inverse kinematic can be solved analytically for some simple manipulators utilising algebraic intuition. The problem can become complicated when dealing with nonlinearities. In these cases, there may exist multiple (finite or infinite), unique or no solutions.
A modern method to solve an inverse kinematic problem poses the question as a numeric optimisation problem. It is convenient to use a non-negative, balanced solution to prevent any one element from overfitting in many solvers. In this case, we can leverage scalar quadratic cost functions.

Given a set of joint angles q, the quadradic state error x–x_e can be concatenated through a scaling matrix K. We can pose this in a common Sequential Quadratic Programming SQP solutions as a total cost. We can also minimise the deviation q from zero with a scaling factor W. A minimal solution to these implies minimum end-effector error and minimum deviation from joint space.

$$(q^\star, x^\star) = \arg\min \lim_{q,x} q^T W q + (x - x_e)^\top K (x - x_e)$$
$$\text{s.t. } x - k(q) = 0$$

given the current configuration and pose the above can be initialised. **(Renton, n.d.)** For the purposes of this project simple trigonometric coordinate transformations between cartesian and cylindrical coordinates is used for inverse kinematics.

10. Control Design Methods

10.1. Motivation

In reality,

- We don't have control over the exact position of an object, we can only apply force to the system to produce movement.
- The exact amount of force required on every part can quickly become a complex problem
- We want to minimize the total energy expenditure to achieve the task
- We want to avoid (where possible) natural constraints of the system.
- We want a reliable method of control.

10.2. LQR Control

Linear quadratic regulator (LQR) is the best feedback gain method to establish a linear system in a closed-loop algorithm. This method was adapted to optimise the 3D Printer's system to get identical output state with the desired reference input. The formulation of this control system was performed by computing the error signal based on the quadratic variable and regulator control in the LQR architecture control. The basic equation of LQR can be seen in **, **, and ** in which x and y represent input and output state, respectively. The symbol A,B, and C are positive matrix variables representing uncertainties on the system, whereas u represent the state input vector. **(Kuantama, Tarca and Tarca, 2018)**

$$\dot{x} = Ax + Bu \tag{10.1}$$
$$y = Cx \tag{10.2}$$
$$u = -kx \tag{10.3}$$

Premise of Optimal Control

Optimal control theory is a division of mathematical optimisation that deals with discovering control for a dynamical system over a period of time such that an objective function is optimised [15].
It is used in numerous applications in science, engineering and operations research. For example, the dynamical system might be a spacecraft with controls corresponding to rocket thrusters, and the objective might be to reach the moon with minimum fuel expenditure [16]. Or the dynamical system could be a nation's economy to minimise unemployment; in this case, the controls could be fiscal and monetary policy [17]. A dynamical system may also be introduced to embed operations research problems within the optimal control theory framework [18].
Linear quadratic regulator is an optimal control regulator that better tracks a reference trajectory compared to a traditional controller such as PID. PID controllers are usually not a choice of use when multi variables systems are in use.

Formulation of a cost function

A cost function, sometimes also known as a loss function; is widely used in mathematical optimisation and decision theory. It's a function that is commonly used to chart an event or values of a particular or multiple variables onto a real number, indicating some costs related to the event.

Premise of Optimal Control

The quadratic cost function is popular due to its useful properties. It is widely used across linear as well as non-linear control this is due to it being convex and smooth. This makes evaluation of derivatives easy. Max is easier to define so that the cost function minimises at a point where the final equilibrium is required. In linear systems, the control design usually translates into some form of Ricatti equation. Ricatti's equations solutions and properties are easy to investigate. While using this in non-linear systems, it is found convenient to be used with Lyapunov functions For quadratic optimal control functions. The popularity of quadratic optimal control function has risen since the whole control engineering area is developed around it, such as LQR and LQG.

Given a continuous time linear system,

$$\dot{x} = Ax + Bu \tag{10.4}$$

It's quadratic cost is given by,

$$J = \int_0^\infty (x^T Q_x x + u^T Q_u u) dt \tag{10.5}$$

Issues arise when deviating from setpoint.

11. Feedback Linearization

The notion of feedback linearization of nonlinear systems is a relatively recent idea in control theory, whose practical realization has been made possible by the rapid development of microprocessor tech-

nology. The basic idea of feedback linearization control is to transform a given nonlinear system into a linear system by use of a nonlinear coordinate transformation and nonlinear feedback. Feedback linearization is a useful paradigm because it allows the extensive body of knowledge from linear systems to be brought to bear to design controllers for nonlinear systems. The roots of feedback linearization in robotics predate the general theoretical development by nearly a decade, going back to the early notion of feedforward-computed torque.

In the robotics context, feedback linearization is known as inverse dynamics. The idea is to exactly compensate all of the coupling nonlinearities in the Lagrangian dynamics in a first stage so that a second-stage compensator may be designed based on a linear and decoupled plant. Any number of techniques may be used in the second stage. The feedback linearization may be accomplished with respect to the joint space coordinates or with respect to the task space coordinates. Feedback linearization may also be used as a basis for force control, such as hybrid control and impedance control.

11.1. Joint Space Inverse Dynamics

We first present the main ideas in joint space where they are easiest to understand. The control architecture we use is important as a basis for later developments. Thus, given the plant model

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau \tag{11.1}$$

we compute the nonlinear feedback control law

$$\tau = M(q)a_q + C(q,\dot{q})\dot{q} + g(q) \tag{11.2}$$

where $a_q \in R^n$ is, as yet, undetermined. Since the inertia matrix M(q) is invertible for all q, the closed-loop system reduces to the decoupled double integrator

$$\ddot{q} = a_q \tag{11.3}$$

Given a joint space trajectory, $q^d(t)$, an obvious choice for the outer loop term aq is as a PD plus feedforward acceleration control.

$$a_q = \ddot{q}^d + K_P(q^d - q) + K_d(\dot{q}^d 0\dot{q}) \tag{11.4}$$

Substituting Equation 11.4 into Equation 11.5 and defining

$$\tilde{q} = q - q^d \tag{11.5}$$

we have the linear and decoupled closed loop system

$$\ddot{\tilde{q}} + K_d\dot{\tilde{q}} + K_P\tilde{q} = 0 \tag{11.6}$$

We can implement the joint space inverse dynamics in a so-called inner loop/outer loop architecture as shown in Fig 4

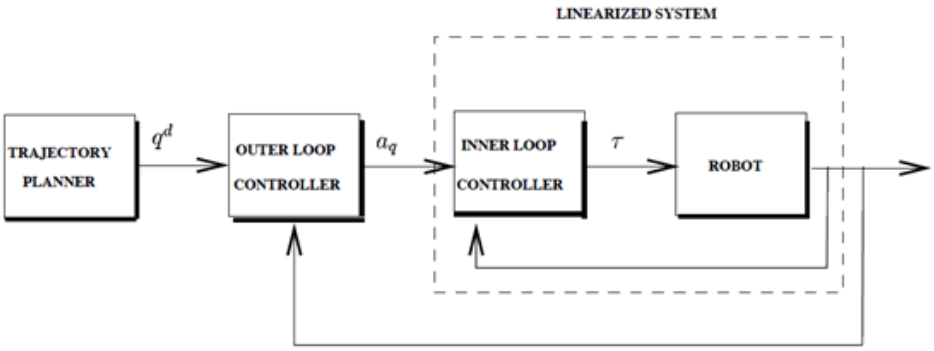

Figure 4: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

The computation of the nonlinear terms in Equation 11.2 is performed in the inner loop, perhaps with a dedicated microprocessor to obtain high computation speed. The computation of the additional term a_q is performed in the outer loop. This separation of the inner loop and outer loop terms is important for several reasons. The structure of the inner loop control is fixed by Lagrange's equations. What control engineers traditionally think of as control system design is contained primarily in the outer loop. The outer loop control given in Equation 11.4 is merely the simplest choice of outer loop control and achieves asymptotic tracking of joint space trajectories in the ideal case of perfect knowledge of the model given by Equation 11.1. However, one has complete freedom to modify the outer loop control to achieve various other goals without the need to modify the dedicated inner loop control. For example, additional compensation terms may be included in the outer loop to enhance the robustness to parametric uncertainty, unmodeled dynamics, and external disturbances. The outer loop control may also be modified to achieve other goals, such as tracking of task space trajectories instead of joint space trajectories, regulating both motion and force, etc. The inner loop/outer loop architecture thus unifies many robot control strategies from the literature. **[5]**

Shortfalls (further work here)

No limits, often output saturation or fine tuning is required

11.2. MPC

Finite horizon optimal control

LQR short term

Use of constraints

We can tell the computer what it can and can't do.

MPC allows us to impose constraints on our system which are a result of physical constraints (actuator output and slew rate limits, displacement limits on the masses) and safety constraints (ensuring the net moment acting on the vertical beam is below a certain threshold). These are imposed as constrains on the states and the outputs of the system. (show math for this)

Optimization

Leveraging advances in SQP programs and theory

NLMPC

Nonlinear model predictive control (henceforth abbreviated as "NMPC" is an optimization-based method for the feedback control of nonlinear systems. Its primary applications are stabilization and tracking problems, which we briefly introduce in order to describe the basic idea of model predictive control. **(Grune and Pannek, 2017)**

Suppose we are given a controlled process whose state $x(n)$ is measured at discrete time instants $t_n, n = 0, 1, 2,$ "Controlled" means that at each time instant we can select a control input $u(n)$ which influences the future behaviour of the state of the system. In tracking control, the task is to determine the control inputs $u(n)$ such that $x(n)$ follows a given reference $x^ref(n)$ as good as possible. This means that if the current state is far away from the reference then we want to control the system towards the reference and if the current state is already close to the reference then we want to keep it there. In order to keep this introduction technically simple, we consider $x(n) \in X = R^d$ and $u(n) \in U = R^m$, furthermore we consider a reference which is constant and equal to $x_* = 0$,i.e.,$x^ref(n) = x_* = 0$for all $n \geq 0$. With such a constant reference, the tracking problem reduces to a stabilization problem; in its full generality the tracking problem will be considered in Section 17.2 **(Grune and Pannek,2017)**

Since we want to be able to react to the current deviation of $x(n)$from the reference value $x_* = 0$, we would like to have $u(\eta)$ in feedback form, i.e, in the form $u(\eta) = \mu(x(n))$ for some map η mapping the state $x \in X$ into the set U of control values. The idea of model predictive control-linear or nonlinear-is now to utilize a model of the process in order to predict and optimize the future system behaviour. A model below of a form is mentioned in 11.7 below.

$$x^+ = f(x, u) \tag{11.7}$$

where $f : X \times U \to X$ is a known and in general nonlinear map which assigns to a state x and a control value u the successor state x^+ at the next time instant. Starting from the current state $x(n)$, for any given control sequence $u(0), \ldots, u(N - I)$ with horizon length $N \geq 2$, we can now iterate (1.1) in order to construct a prediction trajectory x_u defined by

$$x_u(O) = x(n), \quad x_u(k + I) = f\left(x_u(k), u(k)\right), \quad k = 0, \ldots, N - I$$

Proceeding this way, we obtain predictions $x_u(k)$ for the state of the system $x(n + k)$ at time t_{n+k} in the future. Hence, we obtain a prediction of the behavior of the system on the discrete interval $t_n, \ldots, t_{n+N}$ depending on the chosen control sequence $u(0), \ldots, u(N - 1)$

Now we use optimal control in order to determine $u(0), \ldots, u(N-I)$ such that x_u is as close as possible to $x_* = 0$. To this end, we measure the distance between $x_u(k)$ and $x_* = 0$ for $k = 0, \ldots, N-1$ by a function $\mathrm{l}\left(\mathrm{x_u(k)}, \mathrm{u(k)}\right)$. Here, we not only allow for penalizing the deviation of the state from the reference but also if desired, the distance of the control values $u(k)$ to a reference control u_*, which here we also choose as $u_* = 0$. A common and popular choice for this purpose is the quadratic function

$$\mathrm{l}\left(\mathrm{x_u(k)}, \mathrm{u(k)}\right) = |\mathrm{x_u(k)}|^2 + \lambda |u(k)|^2$$

where $||.||$ denotes the usual Euclidean norm and $\lambda \geq 0$ is a weighting parameter for the control, which could also be chosen as 0 if no control penalization is desired. The optimal control problem now reads

$$\mathrm{J(x(n), u(\cdot))} := \sum_{\mathrm{k=0}}^{\mathrm{N-1}} \mathrm{l}\left(\mathrm{x_u(k)}, \mathrm{u(k)}\right)$$

with respect to all admissible I control sequences $\mathrm{u(0), \ldots, u(N-1)}$ with x_u generated by (1.2) Let us assume that this optimal control problem has a solution which is given by the minimizing control sequence $u^*(0), \ldots, u^*(N-I)$, i.e.

$$\min_{u(0),\ldots,u(N-1)} J(x(n), u(\cdot)) = \sum_{k=0}^{N-1} l\left(x_{u^*}(k), u^*(k)\right)$$

In order to get the desired feedback value $\mu(x(n))$, we now set $\mu(x(n)) := u^*(0)$, i.e., we apply the first element of the optimal control sequence. This procedure is sketched in Fig. 1.1 .

At the following time instants t_{n+1}, t_{n+2}, we repeat the procedure with the new measurements $x(n+1), x(n+2)$, in order to derive the feedback values $\mu(x(n+1)), \mu(x(n+2))$, In other words, we obtain the feedback law μ by an iterative online optimization over the predictions generated by our model (1.1).[2] This is the first key feature of model predictive control. (Grüne and Pannels, 2017)

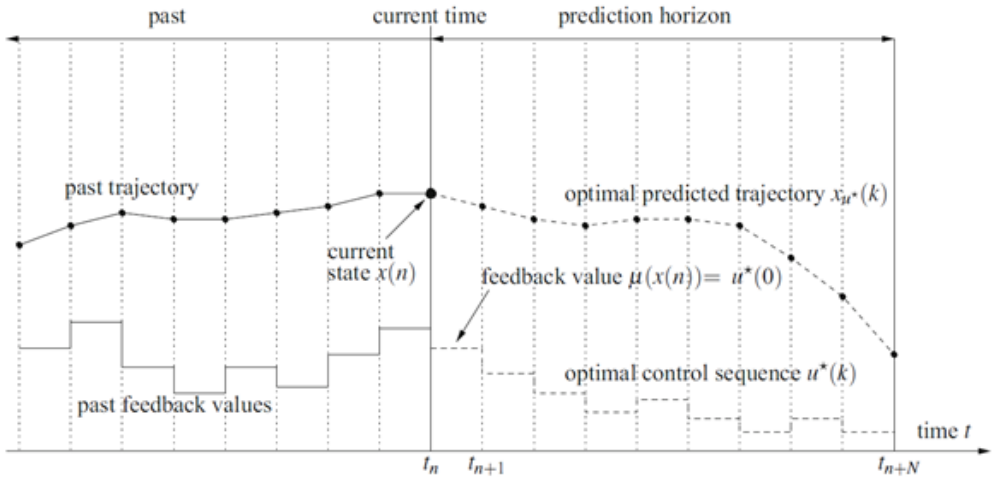

Figure 5: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

Fig. 4 Mustration of the NMPC step at time t_n

From the prediction horizon point of view, proceeding this iterative way the trajectories $x_u(k), k = 0, \ldots, N$ provide a prediction on the discrete interval $t_n, \ldots, t_{n+N}$ at time t_n, on the interval $t_{n+1}, \ldots, t_{n+N+1}$ at time t_{n+l}, on the interval $t_{n+2}, \ldots, t_{n+N+2}$ at time t_{n+3} and so on. Hence, the prediction horizon is moving and this moving horizon is the second key feature of model predictive control. **(Grüne and Pannek, 2017)**

Regarding terminology, another term which is often used alternatively to model predictive control is receding horizon control. While the former expression stresses the use of model-based predictions, the latter emphasizes the moving horizon idea. Despite these slightly different literal meanings, we prefer and follow the common practice to use these names synonymously. The additional term nonlinear indicates that our model 11.7 need not be a linear map. **(Grüne and Rannels 2017)**

Chapter 2

12. System Overview

In your own words, step by step, what does the program do to simulate the system

12.1. Flow diagram

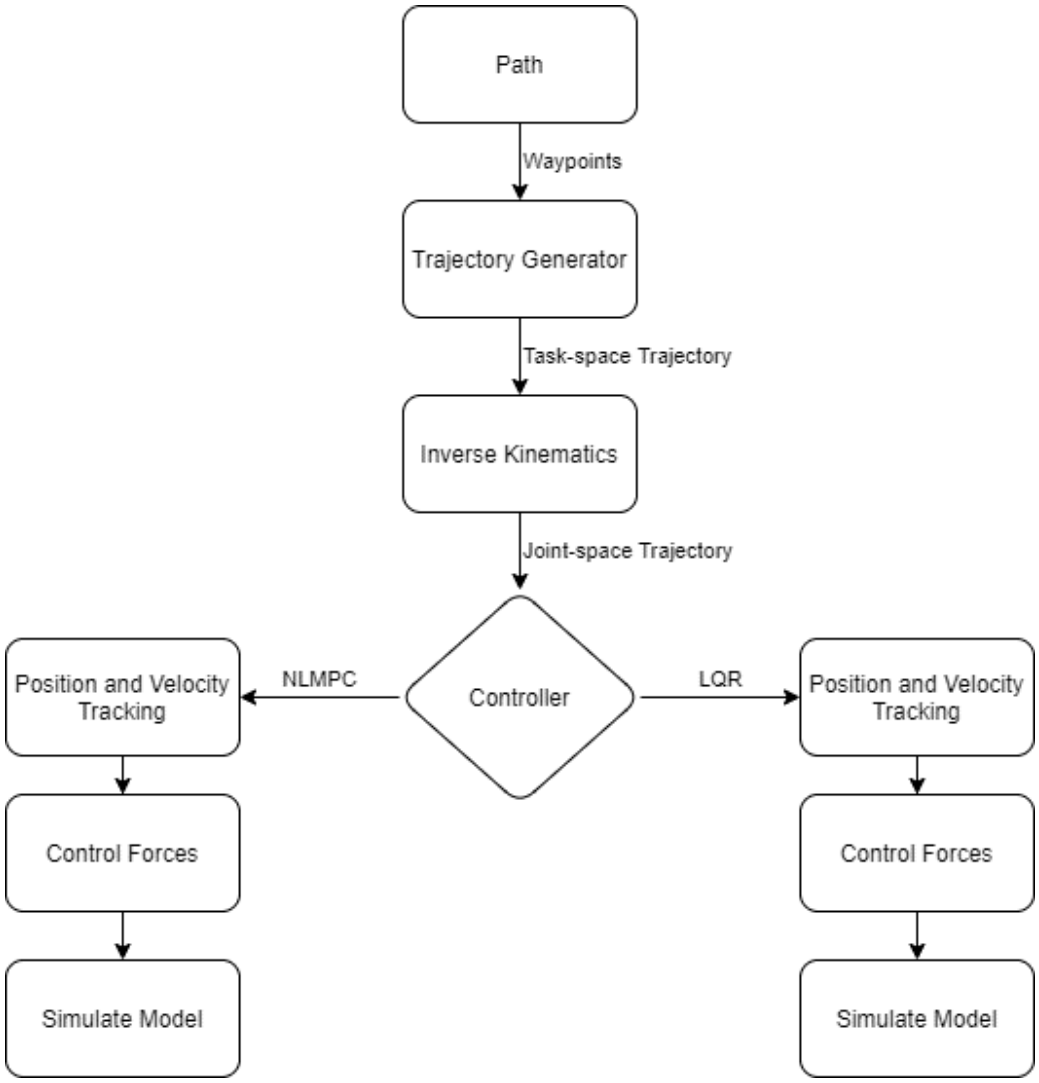

Figure 6: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

13. Crane System Modelling

13.1. Tower Crane System

The tower crane model is based on 6 with key properties shown in table 1. This design

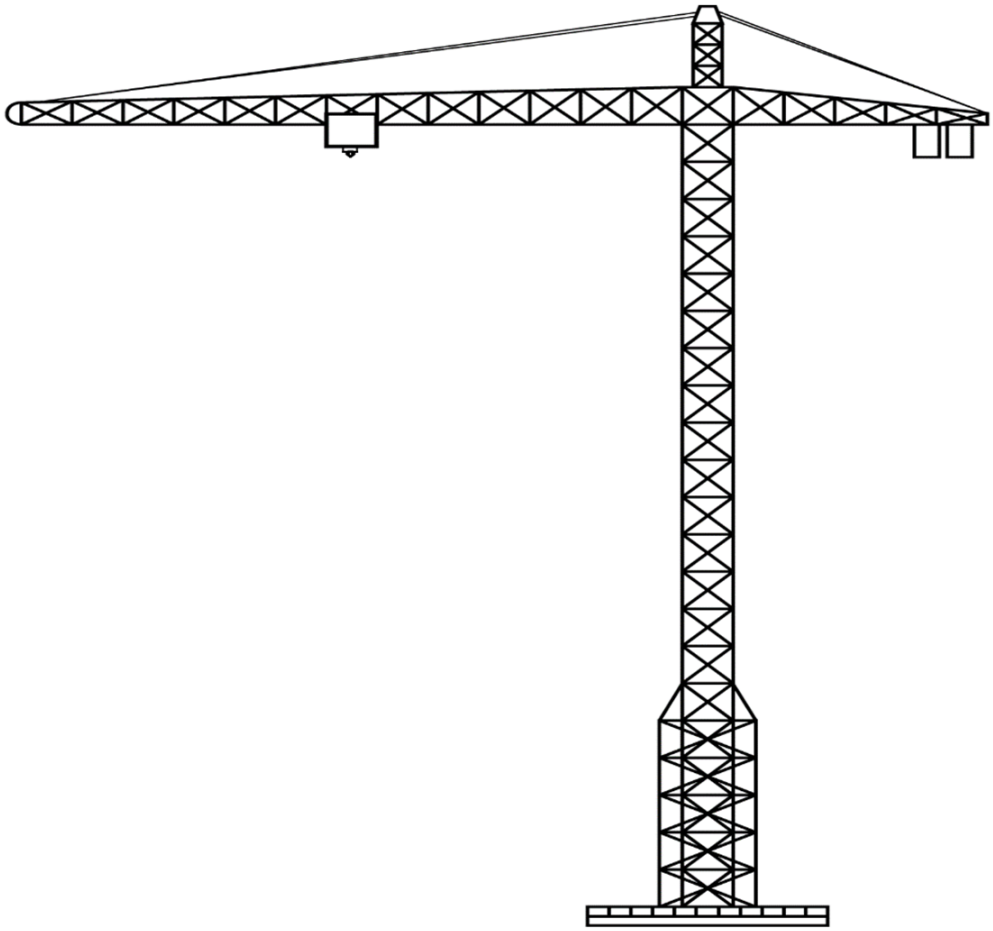

Figure 7: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

Assumptions

- There will be 4 actuators controlling the crane. Actuator 1 and 2 will slide the end effector and counterweight along their respective gantry. Actuator 3 will rotate the crane about the tower and actuator 4 will slide the crane up and down the tower.
- Instead of a hook, the end effector will be a nozzle which deposits filament which is stored externally from the crane and pumped through piping.

Table 1: Table meh

Parameters	Values
End Effector (m_1)	1 kg
Counterweight (m_2)	3.6 kg
Horizontal Beam length	1.4 m
Vertical Beam length	1 m
Mass of Horizontal beam	1.4 kg
Mass of Vertical Beam	5 kg

The table 1 above represents the values of the constable assumed in the modelling and simulation of the project.

13.2. Fully Actuated System

In the study of mechanisms are acquired two concepts very important such as the direct and indirect action. The first consists of movement of elements by action of an actuator, while the second consists of the action of motion transmitted by another interconnected element. Such movements are known as DOF, so that mechanical systems or mechanisms can be classified depending on the number of DOF and the number of actuators. The fully actuated mechanical systems are those having the same number of DOF and actuators.

Underactuated mechanical systems are those with fewer actuators than DOF. It is important to highlight the advantages of underactuated systems, since if they do not have advantages over fully actuated mechanical systems, it will not make sense its development. The main advantages present in underactuated systems are energy saving and control efforts. However, these systems are intended to perform the same functions of fully actuated systems without their disadvantages. (Duarte Madrid et al., 2017)

13.3. Mathematical Model

The tower crane is converted into a kinematic model and states are assigned:

$$q = \begin{bmatrix} r1 \\ r2 \\ \theta \\ z \end{bmatrix} \tag{13.1}$$

Which can be used to describe the motion of mass 1, mass 2, beam 1 and beam 2

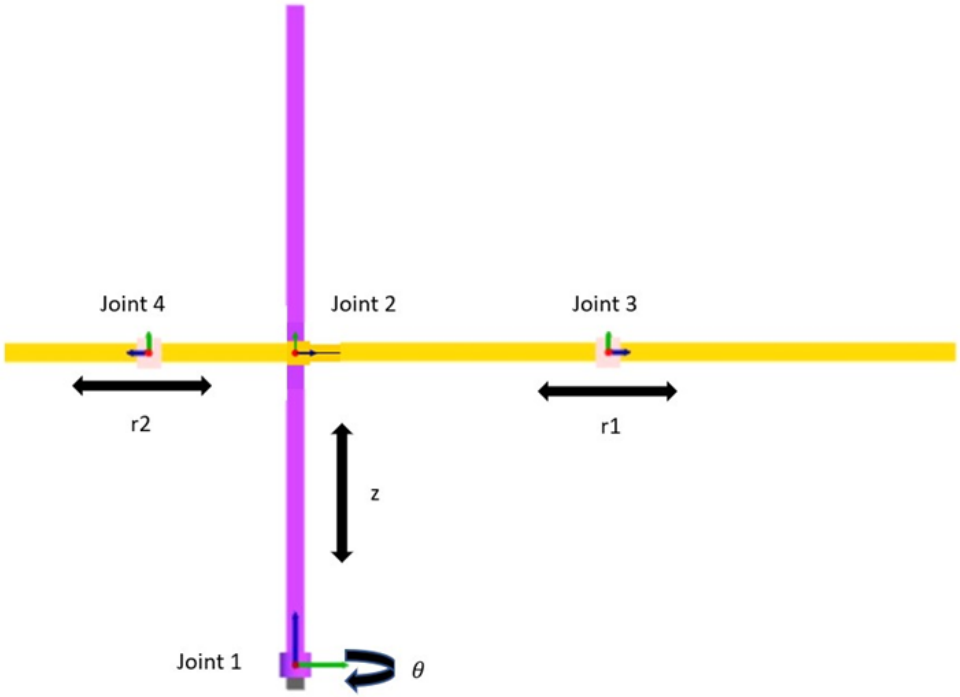

Figure 8: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

13.4. Forward Kinematics of the crane system

As it can be seen in the figure above joint 3 and joint 4 are not in a continuous kinematic chain therefore separate transformations were needed. The table below shows all the parameters that were required to obtain the forward kinematics for joint three and joint 4.

Table 2: DH Parameters table for base to joint 3 and joint 4

Joint #	Joint Type	Joint offset 'd'	Link length 'a'	Joint angle 'θ'	Twist angle'α'
1	Revolute	0	0	0	0
2	Prismatic	z	0	0	-90
3	Prismatic	r2	0	0	0

Joint #	Joint Type	Joint offset 'd'	Link length 'a'	Joint angle 'θ'	Twist angle'α'
1	Revolute	0	0	0	0
2	Prismatic	z	0	0	90
4	Prismatic	r1	0	0	0

Using the DH equation (9.1) mentioned in the background with the identified parameters in Table 2 a homogenous transformation matrix $A_n^($n$-1)$ is produced representing transformation from $n-1$ to n.

$$T_n^0(q) = A_1^0(q_1)A_2^1(q_2)\dots A_n^{n-1}(q_n) \tag{13.2}$$

The general equation for transformation between continuous kinematic chain is represented above.

$$T_3^0(q) = A_1^0(q_1)A_2m1^1(q_2)A_3m1^2m1(q_3) \tag{13.3}$$

$$T_4^0(q) = A_1^0(q_1)A_2m2^1(q_2)A_4m2^2m2(q_4) \tag{13.4}$$

Since we do not have a continuous kinematic chain, we will require two transformation equations as represented above. T_3^0 is for the transformation from origin to r_2 while T_4^0 is the transformation from origin to r_1, where r_1 is considered to be the position of the end effector for this project.
These transformations are beneficial as they are used in the project for converting joint space coordinates to cartesian coordinates which is also used simulation, verification and visualisation purposes.

Post the final A matricies (aka, the numbers inside them at a random position to demonstrate, then post a photo of the crane with the same numbers in each spot.

$$FKM(\begin{bmatrix} r_1, & r_2 \text{ based on } r_1, & \theta, & z \end{bmatrix})$$

FKM([.5,(-0.08+.5+0.5)/3.6,pi/2,.5])

$$A_0^1 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_{2m2}^1 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & -1 & 0 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_0^1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_0^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_{2m2}^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.2556 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_0^1 = \begin{bmatrix} 0 & 0 & 1 & 0.5 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_0^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_{2m2}^1 = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.2556 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_0^1 = \begin{bmatrix} 0 & 0 & 1 & 0.5 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

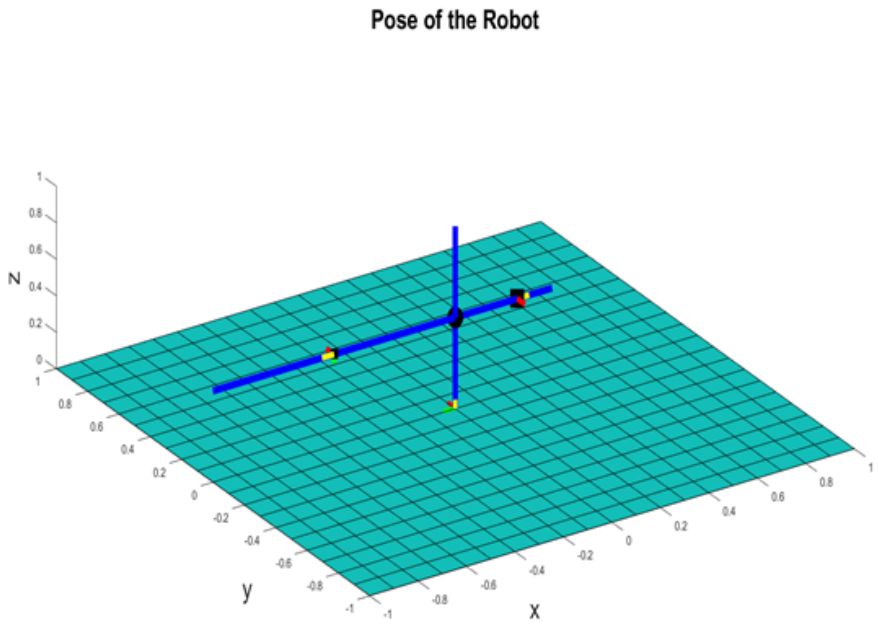

Figure 9: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

13.5. Inverse kinematics of crane system

Using simple cartesian to polar coordinate transformation following equations are obtained as listed below this also includes calculation for position of r_2 making $\sum$ Moments $= 0$

$$r_1 = \sqrt{x^2 + y^2} \tag{13.5}$$

$$\theta = \arctan\left|\frac{y}{x}\right| \tag{13.6}$$

$$r_2 = \frac{beam\,\text{torque} + m_1 * r_1}{m_2} \tag{13.7}$$

$$z = z \tag{13.8}$$

14. Modelling a crane system using Lagrangian

In order to calculate the Lagrangian of a system, we compute all the potential energies which are then subtracted from the kinetic energies of the system.

$$\mathcal{L} = T - V$$

$$\mathcal{L}(\boldsymbol{q(t)}, \dot{\boldsymbol{q}}(\boldsymbol{t})) = T(\boldsymbol{q(t)}, \dot{\boldsymbol{q}}(\boldsymbol{t})) - \boldsymbol{V}(\boldsymbol{q(t)})$$

$$\mathcal{L} = \tfrac{1}{2}\left(\boldsymbol{m}_1\boldsymbol{r}_1 + \boldsymbol{m}_2\dot{\boldsymbol{r}}_2^2 + (\boldsymbol{m}_1 + \boldsymbol{m}_2 + \boldsymbol{m}_b)\,\dot{z}^2 + \boldsymbol{m}_1\boldsymbol{r}_1^2\dot{\boldsymbol{\theta}}^2 + \boldsymbol{m}_2\boldsymbol{r}_2^2\dot{\boldsymbol{\theta}}^2 + \boldsymbol{I}_b\dot{\boldsymbol{\theta}}^2\right) - g\mathbf{z}\,(\boldsymbol{m}_1 + \boldsymbol{m}_2 + \boldsymbol{m_b})$$

Where m_1 is mass of joint $3, m_2$ is mass of joint $4, m_b$ is mass of the horizontal beam and

$$I_b = \left(\frac{1}{12} m l_{cm}^2 + md^2\right) + \left(\frac{1}{2} m r^2\right), \text{ using paralled axis theorem}$$

Listed above are all the kinetic and potential energies taken into account which are linear motion of m along the beam (nozzle), linear motion of m_2 along the beam, linear motion of all masses along z axis, angular motion of m_l, angular motion of m_2, angular motion of beam and potential energy due to gravity of all the masses respectively.
We now calculate Euler-Lagrange equations using the following:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}(q,\dot{q})}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}(q,\dot{q})}{\partial q_i} = \tau_i$$

$$\frac{\frac{m_1 \dot{r}_1}{m_2 \dot{r}_2}}{\frac{d}{dt}} \begin{bmatrix} m_1 r_1 \dot{\theta}^2 \\ m_2 r_2 \theta^2 \\ 0 \\ -\left(m_1 + m_2 + m_b\right) g \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ \tau \\ F_z \end{bmatrix}$$

$$\left(m_1 + m_2 + m_b\right) \dot{z}^2 \dot{\theta} + m_2 r_1^2 \dot{\theta} + I_b \dot{\theta}\Big] - \begin{bmatrix} F_1 \\ F_2 \\ \tau \\ F_z \end{bmatrix}$$

Second order equations listed below

$$\ddot{\mathrm{r}}_1 = \frac{\mathrm{F}_1 + \mathrm{m}_1 \mathrm{r}_1 \dot{\theta}^2}{\mathrm{m}_1}$$

$$\ddot{\mathrm{r}}_2 = \frac{\mathrm{F}_2 + \mathrm{m}_2 \mathrm{r}_2 \dot{\theta}^2}{\mathrm{m}_2}$$
$$\ddot{\theta} = \frac{\tau - 2(\ \mathrm{m}_1 \mathrm{r}_1 \dot{\mathrm{r}}_1 + \mathrm{m}_2 \mathrm{r}_2 \dot{\mathrm{r}}_2)\dot{\theta}}{\mathrm{m}_1 \mathrm{r}_1^2 + \mathrm{m}_2 \mathrm{r}_2^2 + \mathrm{I}_\mathrm{b}}$$
$$\ddot{\mathrm{z}} = \frac{\mathrm{F}_\mathrm{z} - (\mathrm{m}_1 + \mathrm{m}_2 + \mathrm{m}_\mathrm{b})\mathrm{g}}{(\mathrm{m}_1 + \mathrm{m}_2 + \mathrm{m}_\mathrm{b})}$$

Dynamic equation of an n-link robot in the matrix form is listed below

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = u$$

Where

$$M(\boldsymbol{q})\ddot{q} = \begin{bmatrix} m_1 & 0 & 0 & 0 \\ 0 & m_2 & 0 & 0 \\ 0 & 0 & m_1 r_1^2 + m_2 r_2^2 + I_b & 0 \\ 0 & 0 & 0 & m_1 + m_2 + m_b \end{bmatrix} \begin{bmatrix} \ddot{r}_1 \\ \ddot{r}_2 \\ \ddot{\theta} \\ \ddot{z} \end{bmatrix}$$

$$C(q,\dot{q})\dot{q} = \begin{bmatrix} 0 & 0 & -m_1 r_1 \dot{\theta} & 0 \\ 0 & 0 & -m_2 r_2 \theta & 0 \\ 2m_1 r_1 \dot{\theta} & 2m_2 r_2 \dot{\theta} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ \theta \\ \dot{z} \end{bmatrix}, \quad g(q) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ g\left(m_1 + m_2 + m_b\right) \end{bmatrix} u = \begin{bmatrix} F_1 \\ F_2 \\ \tau \\ F_z \end{bmatrix}$$

Chapter 3

15. Motion control and Trajectory Generation

Given a set of wav-points that describes the build order of a set of walls, we need a reference to track that is achievable by the machinery. This is a non-trivial problem for a few reasons.

15.1. Trapezoidal Model of Speed

In 3D extrusion printing, there is an optimum speed at which the printer head can lay down the ideal thickness of material. On any run of continuous printing, it is preferred that the head starts from rest but accelerates at a reasonable rate to this optimum speed which is maintained until near the end of the run when deceleration returns the head to rest. In the simplest case of a run along a straight-line segment, the acceleration and deceleration will be constant and of the same size giving a trapezoidal speed profile. This trapezoidal profile is illustrated in Fig. 5.

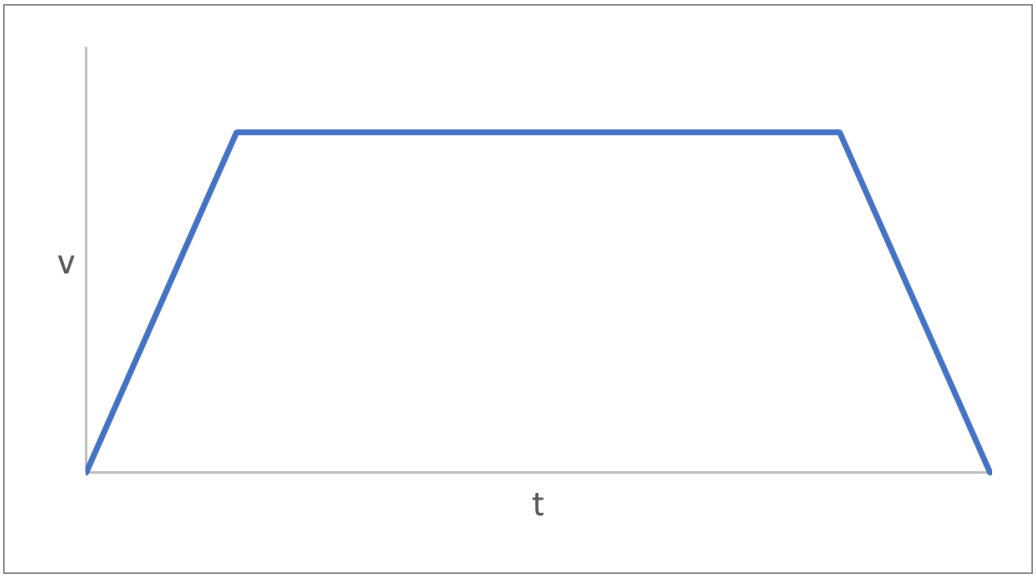

Figure 10: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

Fig. 7 Trapezoidal profile of speed (v) against time (t With time variable t, distance travelled s, velocity $v = \frac{ds}{dt}$ m/s, preferred acceleration rate a, optimum head speed u and total distance to travel d, the requirements are that:

- The run starts at rest at time $t = 0$ with no distance travelled so $s(0) = t(0) = 0$
- The head accelerates with constant acceleration a until time t_1 when speed is u so $v\left(t_1\right) = u$
- The velocity is held constant until time t
- The head decelerates at a constant rate until it comes to rest at time t_e so $v\left(t_\theta\right) = 0$
- The total distance travelled is d so $s\left(t_e\right) = a$

Then the dynamic system is specified by:

$$\frac{dv}{dt} = \begin{cases} a & 0 \leq t \leq t_1 \\ 0 & t_1 < t \leq t_2 \\ -a & t_2 < t \leq t_e \end{cases}$$

Assuming that, with the constant acceleration, the distance to be travelled is long enough to allow the optimum speed to be reached, the speed and distance travelled are then given by:

$$v = \begin{cases} \mathrm{at} & 0 \leq \mathrm{t} \leq \mathrm{t}_1 \\ \mathrm{u} & \mathrm{t}_1 < \mathrm{t} \leq \mathrm{t}_2 \\ \mathrm{u} - \mathrm{a}\left(\mathrm{t} - \mathrm{t}_2\right) & \mathrm{t}_2 < \mathrm{t} \leq \mathrm{t}_\mathrm{e} \end{cases}$$

$$s = \begin{cases} \mathrm{at}^2/2 & 0 \leq \mathrm{t} \leq \mathrm{t}_1 \\ \mathrm{u}\left(\mathrm{t} - \mathrm{t}_1\right) + \frac{\mathrm{u}^2}{2\mathrm{a}} & \mathrm{t}_1 < \mathrm{t} \leq \mathrm{t}_2 \\ \mathrm{u}\left(\mathrm{t} - \mathrm{t}_1\right) - \mathrm{a}\left(\mathrm{t} - \mathrm{t}_2\right)^2/2 + \frac{\mathrm{u}^2}{2\mathrm{a}} & \mathrm{t}_2 < \mathrm{t} \leq \mathrm{t}_\mathrm{e} \end{cases}$$

With $v\left(\mathrm{t}_1\right) = a\mathrm{t}_1 = u, \mathrm{t}_1 = u/a$. To ensure the optimum speed can be reached, the distance travelled at the end of acceleration must be less than half the total distance. That is

$$s\left(t_1\right) = \frac{at_1^2}{2} = \frac{u^2}{2a} < \frac{d}{2}$$

Therefore, this condition can be expressed as $\mathrm{d} > \frac{\mathrm{u}^2}{\mathrm{a}}$. By symmetry, $\mathrm{t_e} - \mathrm{t_2} = \mathrm{t_1} = \frac{\mathrm{u}}{\mathrm{a}}$ so

$$s\left(t_e\right) = ut_e - \frac{u^2}{a} - \frac{a\left(\frac{u}{a}\right)^2}{2} + \frac{u^2}{2a} = ut_e - \frac{u^2}{a} = d$$

to give $t_e = \frac{d}{u} + \frac{u}{a}$ and $t_2 = \frac{d}{u}$ In practice, any continuous extrusion run will, at best, consist of a sequence of straight-line segments of reasonable length allowing optimum speed to be reached on each segment. However, there is an issue of balancing the desire for constant speed against maintaining rigid adherence to the linear sections. The generation of curves requires approximation by many very small line segments where the above model is not at all suitable. To deal with the most general cases, more refined approaches are needed which includes smooth motion that uses the final velocity and acceleration as the initial values for the following segment thus maintaining the velocity.

15.2. Multi-segment Multi-Axis Trajectory

Multi-segment multi-axis trajectory is a way of generating a trajectory based on via points. In the use of robotics there is usually a requirement to move smoothly along a path through one or more via points without stopping. It is a requirement for this project to have a continuous trajectory that involves not stopping at the points this is due to the continuous extrusion from the nozzle of the printer. It could cause an inconsistency on the texture of the extrusion.
To make the extrusion more uniform the velocity has to be constrained to suit the robot's actuators slew rates and the maximum extrusion limit. The path then compromises of linear segments with fifth order quintic polynomials. The trajectory is generated by using the via points with velocity and acceleration limits and a position and a time is recorded for each time segment to suit the frequency of the controller action.

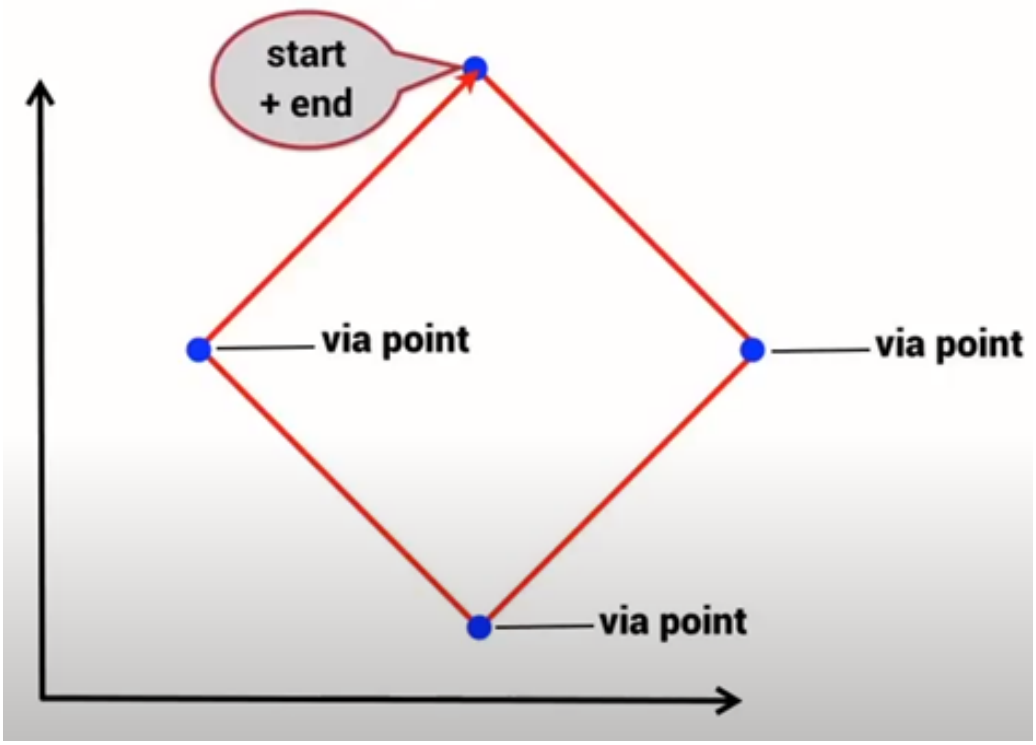

Figure 11: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

The 5 th order quintic polynomial is used as it has six coefficients that allows it to meet its six boundary conditions which are initial and final positions, velocities and accelerations. These polynomials are given by the following equations below.

$$
\begin{gathered}
q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \\
q_0 = a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 + a_4 t_0^4 + a_5 t_0^5 \\
v_0 = a_1 + 2a_2 t_0 + 3a_3 t_0^2 + 4a_4 t_0^3 + 5a_5 t_0^4 \\
\alpha_0 = 2a_2 + 6a_3 t_0 + 12a_4 t_0^2 + 20a_5 t_0^3 \\
q_f = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 + a_4 t_f^4 + a_5 t_f^5 \\
v_f = a_1 + 2a_2 t_f + 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4 \\
\alpha_f = 2a_2 + 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^3 \\
q_0 = \text{ initial position} \\
v_0 = \text{ initial velocity} \\
\alpha_0 = \text{ initial acceleration}
\end{gathered}
$$

where,

$$
\begin{gathered}
q_f = \text{ final position} \\
v_f = \text{ final velocity} \\
\alpha_f = \text{ final acceleration}
\end{gathered}
$$

15.3. Blend feature

The trajectory starts from q_1 at rest and finishes at a point q_M and comes to a stop. However, for all the point in between q_1 and q_M its moves through or closer to the intermediate configurations

in a continuous motion. The over constraints the problem and in order to achieve the intermediate configuration with continuous motion, the ability to reach each intermediate configuration exactly, is surrendered. To make it easier to understand a one-dimensional case is shown in Fig. 7

The robot starts from q_1 at rest and finishes at q_M at rest, but moves through (or close to) the intermediate configurations without stopping. The problem is over constrained and in order to attain continuous velocity we surrender the ability to reach each intermediate configuration. This is easiest to understand for the 1 -dimensional case shown in Fig. 3.5. The motion comprises linear motion segments with polynomial blends, but here we choose quintic polynomials because they are able to match boundary conditions on position, velocity and acceleration at their start and end points.

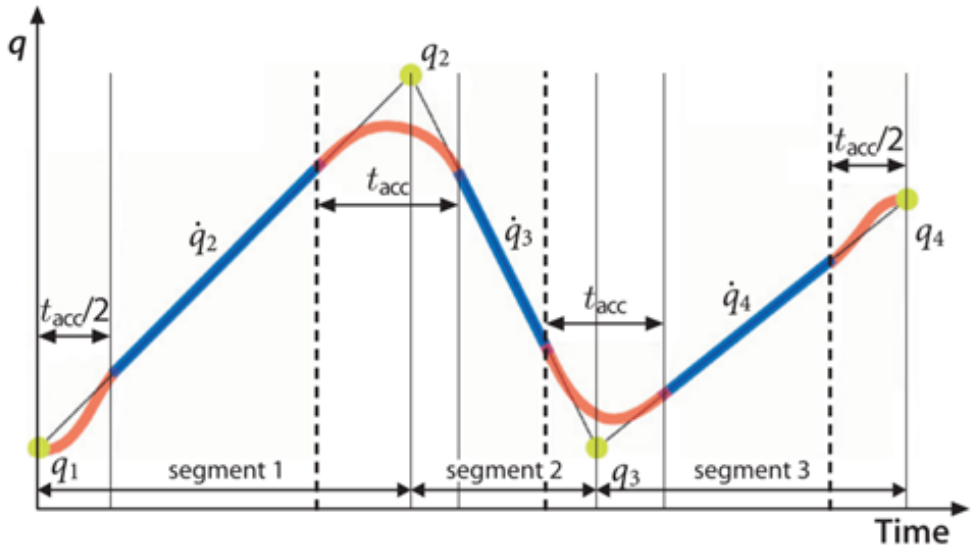

Figure 12: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

The first segment of the trajectory accelerates from the initial configuration q_1 and zero velocity, and joins the line heading toward the second configuration q_2. The blend time is set to be a constant t_{acc} and t_{acc} /2 before reaching q_2 the trajectory executes a polynomial blend, of duration t_{acc} , onto the line from q_2 to q_3, and the process repeats. The constant velocity q_k can be specified for each segment. The average acceleration during the blend is

$$\ddot{q} = \frac{\dot{q}_{k+1} - \dot{q}_k}{t_{acc}}$$

16. Linear Quadratic Regulator (LQR) Control

The LQR requires a linear model to be converted into a state-space model mentioned below

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

In the modelling section we have calculated 4 Euler Lagrange egyations's of motion. These are second order nonlinear equations. For us to be able to control the system with LQR we need to convert the

equations into first order and then use a technique called feedback linearisation to linearize the model. Second order differential equations:

$$\begin{aligned}
\ddot{\mathrm{r}}_1 &= \frac{\mathrm{F}_1+\mathrm{m}_1\mathrm{r}_1\dot{\theta}^2}{\mathrm{m}_1} \\
\ddot{r}_2 &= \frac{\boldsymbol{F}_2+\boldsymbol{m}_2\boldsymbol{r}_2\dot{\theta}^2}{\boldsymbol{m}_2} \\
\ddot{\boldsymbol{\theta}} &= \frac{\boldsymbol{\tau}-2(\boldsymbol{m}_1\boldsymbol{r}_1\dot{\boldsymbol{r}}_1+\boldsymbol{m}_2\boldsymbol{r}_2\dot{\boldsymbol{r}}_2)\dot{\boldsymbol{\theta}}}{\boldsymbol{m}_1\boldsymbol{r}_1^2+\boldsymbol{m}_2\boldsymbol{r}_2^2+\boldsymbol{I}_b} \\
\ddot{\mathbf{z}} &= \frac{\boldsymbol{F}_z-(\boldsymbol{m}_1+\boldsymbol{m}_2+\boldsymbol{m}_b)g}{(\boldsymbol{m}_1+\boldsymbol{m}_2+\boldsymbol{m}_b)}
\end{aligned}$$

For ease of calculation and simplification we have kept the first 4 states as joint positions and the next four states as joint velocities.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \theta \\ z \\ \dot{r}_1 \\ \dot{r}_2 \\ \dot{\theta} \\ \dot{z} \end{bmatrix}$$

The model below is not represented by first order differential equations

$$\begin{bmatrix} \mathrm{x}_5 \\ \mathrm{x}_6 \\ \mathrm{x}_7 \\ \mathrm{x}_8 \\ \dot{\mathrm{x}}_1 \\ \dot{\mathrm{x}}_2 \\ \dot{\mathrm{x}}_3 \\ \dot{\mathrm{x}}_4 \\ \dot{\mathrm{x}}_5 \\ \dot{\mathrm{x}}_6 \\ \dot{\mathrm{x}}_7 \\ \dot{\mathrm{x}}_8 \end{bmatrix} = \begin{bmatrix} \mathrm{r}_1 \\ \dot{\mathrm{r}}_2 \\ \dot{\theta} \\ \dot{\mathrm{z}} \\ \ddot{\mathrm{r}}_1 \\ \ddot{\mathrm{r}}_2 \\ \ddot{\theta} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{F}_1+\mathrm{m}_1\mathrm{x}_1\mathrm{x}_7^2}{\mathrm{m}_1} \\ \frac{\mathrm{F}_2+\mathrm{m}_2\mathrm{x}_2\mathrm{x}_7^2}{\mathrm{m}_2} \\ \frac{\mathrm{F}_z-(\mathrm{m}_1+\mathrm{m}_2+\mathrm{m}_b)\mathrm{g}}{\mathrm{m}_1+\mathrm{m}_2+\mathrm{m}_b} \end{bmatrix}$$

We have used a nonlinear system and obtained a linear system while substituting for the nonlinear terms in the system and adding them back after processing it through the Ricatti equation. The new linear system along with the substituted variables are listed below. Given the above nonlinear system, it needs to be converted to a linear system to be able to be used in a LQR controller. C matrix was chosen since we have 4 encoders in the design which measure our states directly.

$$\begin{aligned}
\dot{x} &= Ax + Bu \\
y &= Cx + Du
\end{aligned}$$

The system above has been converted to 1 st order from a second order set of equations.

$$\text{where } \begin{bmatrix} F_1 \\ F_2 \\ \tau \\ F_z \end{bmatrix} = u$$

As our system does not have one set point or equilibrium point, we will need to use a technique called feedback linearisation. This prevents the model from breaking down as it deviates from the zero point.

17. Feedback Linearization

Feedback linearisation is accomplished by subtracting non-linear terms out of the system equations and adding them up to the control, this results in a linear system provided that the computer executing the control has adequate capability to compute the non-linear terms fast enough and it does not result in the actuators to oversaturate which in our case are considered ideal. The process of this is represented by the equations below. (Franklin et al., 2015)

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = \begin{bmatrix} x_5 \\ x_6 \\ x_7 \\ x_8 \\ \frac{m_1 x_1 x_7^2}{m_1} \\ \frac{m_2 x_2 x_7^2}{m_2} \\ -\frac{2(m_1 x_1 x_5 + m_2 x_2 x_6) x_7}{m_1 x_1^2 + m_2 x_2^2 + I_b} \\ \frac{(m_1 m_2 m_b) g}{(m_1 m_2 m_b)} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m_1} & & & \\ 0 & \frac{1}{m_2} & 0 & 0 \\ 0 & 0 & \frac{1}{m_1 x_1^2 + m_2 x_2^2 + I_b} & 0 \\ 0 & 0 & 0 & \frac{1}{m_1 + m_2 + m_b} \end{bmatrix} \begin{bmatrix} F_1 \\ F - 2 \\ \tau \\ F_z \end{bmatrix} \tag{17.1}$$

The feedback linearizing control is found by simply inspecting above equation a:

$$u = \begin{bmatrix} F_1 \\ F_2 \\ \tau \\ F_z \end{bmatrix} = \begin{bmatrix} \mu_1 - m_1 x_1 x_7^2 \\ \mu_2 - m_2 x_2 x_7^2 \\ \mu_\tau \left(m_1 x_1^2 + m_2 x_2^2 + I_b \right) + 2 \left(m_1 x_1 x_5 + m_2 x_2 x_6 \right) x_7 \\ \left(\mu_4 + g \right) \left(m_1 + m_2 + m_b \right) \end{bmatrix}$$

Substituting 14.2 into 14.1 gives us which is in the State-Space form

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \\ \dot{x}_7 \\ \dot{x}_8 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ m_1 & & & \\ 0 & \frac{1}{m_2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_\tau \\ \mu_4 \end{bmatrix}$$

where

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m_1} & & & \\ 0 & \frac{1}{m_2} & 0 & 0 \\ 0 & 0 & \frac{1}{m_1 x_1^2 + m_2 x_2^2 + I_b} & 0 \\ 0 & 0 & 0 & \frac{1}{m_1 + m_2 + m_b} \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad D = 0$$

It is also worthy to note that the linearised system can be demonstrated to be controllable and observable.

17.1. Controllability and Observability

Controllability and observerability are the two major concepts open modern control system theory. R Kalman in 1960 introduced these concepts. Controllability is the ability of a particular actuator configuration to control all the states of a system while observerability is the ability of a particular sensor configuration to supply all the information necessary to estimate all the states of the system. It is possible to check if a system is controllable using the controllability matrix below and verifying if the rank($\mathcal{C}$) = n, where n is the length of the matrix.

$$\mathcal{C} = \begin{bmatrix} \mathrm{B} & \mathrm{AB} & \mathrm{A}^2\,\mathrm{B} & \cdots & \mathrm{A}^{\mathrm{n}-1}\,\mathrm{B} \end{bmatrix}$$

It is confirmed that the controllability matrix of linearized system is full rank. Similarly, observerability can be verified using the observability matrix mentioned below if rank(O) = n, where n is the length of the matrix.

$$O = \begin{bmatrix} \mathrm{C} \\ \mathrm{CA} \\ \mathrm{CA}^2 \\ \vdots \\ \mathrm{CA}^{\mathrm{n}-1} \end{bmatrix}$$

The observability matrix of the linear system is also full rank, therefore making the system observable.

17.2. Tracking Control Design

Given the linear system

$$\dot{x} = Ax + Bu$$

And a desired trajectory

$$x^*(t)$$
$$u = -kx$$

we can define the error $\tilde{x}(t)$ between our current and desired state at each time t as

$$\tilde{x}(t) = x(t) - x^*(t)$$

with our goal being to find the control action u to minimise this difference at each timestep. We can take the derivative of equation XX to give:

$$\tilde{\boldsymbol{x}}(\boldsymbol{t}) = \dot{\boldsymbol{x}}(\boldsymbol{t}) - \dot{\boldsymbol{x}}^*(\boldsymbol{t})$$

And substituting in equation XX gives:

$$\tilde{\dot{x}} = Ax + Bu - \dot{x}^*(t)$$

Rearranging and substituting in equation XX gives us:

$$\tilde{x} = A\left(\tilde{x} + x^*\right) + Bu - \dot{x}^*(t)$$
$$\dot{\tilde{x}} = A\tilde{x} + Bu + A\dot{x}^*(t) - \dot{x}^*(t)$$

Now, if we rewrite our desired control force μ as:

$$\mu = v_1 + v_2$$

Then we can expand equation XX as:

$$\dot{\tilde{x}} = A\tilde{x} + Bv_2 + Bv_1 - Ax^* + \dot{x}^*(t)$$

Now if we impose that:

$$\boldsymbol{B}v_1 - Ax^* + \dot{x}^*(t) = 0$$

Which can be rearranged to give:

$$v_1 = B^{-1}\left(-Ax^* + \dot{x}^*(t)\right)$$

Equation XX now simplifies to:

$$\dot{\tilde{x}} = A\tilde{x} + Bv_2$$

And we can use a simple gain k from our LQR such that:

$$v_2 = -k\tilde{x} = -k\left(x - x^*(t)\right)$$

Thus, we now have a position and velocity tracking controller for our linearised system.

18. Cost function weightings and computed gains

The cost function is shown in eq (14.1) below.

$$C = \int_0^\infty \left(x^T Q_x x + u^T Q_u u\right) dt$$

where

$$Q_u = \begin{bmatrix} 2000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 200 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 200 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix} \tag{18.1}$$

$$Q_u = 0.001 \tag{18.2}$$

The costs are chosen to prioritise the position states in Q_x. It is our priority to achieve the desired positions for increased precision and repeatability. The velocity states in Q_x are punished as they could have a little flexibility compared to the positions. The input cost Q_u is punished for reducing the amount of force generated as a control action

The gains K in eq (14.6) are computed using the linearized system and processing it through the Algebraic Ricatti equation (14.4) along with the cost function.

$$A^T S + SA - SBQ_u^{-1}B^T S + Q_x B^T + Q_x = 0$$

Where K is derived from S using eq (14.5)

$$K = R^{-1}\left(B^T S + N^T\right)$$

Thus, the K gains are computed as

$$K = \begin{bmatrix} 1414.2 & 0.0000 & 0.0000 & 0.0000 & 450.40 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 1414.2 & 0.0000 & 0.0000 & 0.0000 & 458.50 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 1000.0 & 0.0000 & 0.0000 & 0.0000 & 319.40 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 316.20 & 0.0000 & 0.0000 & 0.0000 & 103.10 \end{bmatrix} \tag{18.3}$$

19. Non-linear Model Predictive Control NLMPC

19.1. Fmincon

Fmincon finds a minimum of a constrained nonlinear multivariable function.

$$\min_x f(x) \text{ such that } \begin{cases} c(x) \leq 0 \\ \text{ceq}(x) = 0 \\ A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb < x < ub \end{cases}$$

b and beq are vectors, A and Aeg are matrices, $c(x)$ and ceg(x) are functions that return vectors, and $f(x)$ is a function that returns a scalar. $f(x), c(x)$, and ceg(x) can be nonlinear functions. $x, /b$, and yb can be passed as vectors or matrices [**9**]

fmincon finds a constrained minimum of a scalar function of several variables starting at an initial estimate. This is generally referred to as constrained non-linear optimisation or non-linear programming **[9]**.

19.2. Sequential Quadratic Programming (SQP)

SQP methods represent state-of-the-art in nonlinear programming methods. Schittowski [34], for example, has implemented and tested a version that out performs every other tested method in terms of efficiency, accuracy, and percentage of successful solutions, over a large number of test problems. Based on the work of Biggs [1], Han [22], and Powell ([31],[32]), the method allows you to closely mimic Newton's method for constrained optimization just as is done for unconstrained optimization. At each major iteration an approximation is made of the Hessian of the Lagrangian function using a quasi- Newton updating method. This is then used to generate a QP subproblem whose solution is used to form a search direction for a line search procedure. An overview of SQP is found in Fletcher [13], Gill et al. [19] , Powell [33] , and Schittowski [23] . The general method, however, is stated here.

Given the problem description in GP (Eq. 2.1) the principal idea is the formulation of a QP subproblem based on a quadratic approximation of the Lagrangian function

$$L(x, \lambda) = f(x) + \sum_{i=1} \lambda_i \cdot g_i(x)$$

Here Eq. 2.1 is simplified by assuming that bound constraints have been expressed as inequality constraints. The QP subproblem is obtained by linearizing the nonlinear constraints. (Computation Visualization Programming For Use with MATLAB ® User's Guide Optimization Toolbox, 2001)

Code Flow

The simulation code works by initialising the parameters of the system and then generating the trajectory using our provided path. We then run a loop to simulate the system whereby we generate a control action based on the current states of the system, then predict using our model the states of the system at the next time step. This prediction can then be used at the next time step and the loop repeats over the course of our entire trajectory.
We then plotted the results to ensure the trajectory was matched.

Cost Function

The cost function is shown below

```
function cost = costfunction(U,qstar, q0,H,simstep)

cost = 0; %Initialise nil cost

Q = diag([2e5 2e5 1e5 0.1e5 2e1 2e1 1e1 0.1e1]);    %State Weights matrix
R = diag([1 1 1 1]);                                %Control-input Weights matrix

q = repmat(q0(:,1),1,H+1);                         %Pre-allocate size for faster execution

for i = 1:H

    q(:,i+1)= CraneModel(q(:,i),U(:,i));      %Compute dx = fx + gu based on current x and u over horizon H
    cost = cost + ((qstar(:,simstep -1 + i) - q(:,i))'*Q*(qstar(:,simstep -1 + i) - q(:,i)));

end
end
```

Figure 13: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

Give an example in maths notation of a single cost being generated, for one horizon step, Then give an example of what the cost would be Fig. 10:The cost function utilised by fmincon for MPC The objective cost functions goal was to find such a control force that minimised the error between the end effector position and its desired position at each step in time along its trajectory. This was done by computing the 8xH horizon prediction matrix qH based on the nominal 4xH horizon control matrix U. The cost was then defined as the sum of the squared error between the current and desired states at each point along the trajectory multiplied by the tuning matrix Q. This framework mimics a simple Proportional controller.
As we assumed ideal actuators, there was no cost punishment on the actuators.

Constraints and Bounds

Upper-bound and lower-bound for control forces U based on actuator torque limits.

Chapter 4

20. Simulation

20.1. MATLAB functions for Simulated System

Function Name	Description
FKM.m	Generates the transformation matrices given joint angle and displacements
IKM.m	Generated the joint angles and displacements given cartesian coordinates.
costvel.m	Cost function used for NLMPC.
massmatrix.m	Run the simulation for the nonlinear model using ode4
nonlcon.m	Non-linear constraints for the NLMPC
ode4.m	4th order Runga-Kutta solver
TrajectoryGenereration.m	Generates a trajectory as a trapezoidal profile between each waypoint.
TrajectoryGenerationNEW.m	Generates a trajectory given a velocity and acceleration constraint while incorporating the blend feature for smooth motion.
RUNLQR.m	Run a script with a set of instructions for LQR control and Feedback linearization to simulate the model given some waypoints.
RUNMPC.m	Run a script with a set of instructions for MPC control and Feedback linearization to simulate the model given some waypoints.
Parameters.m	Parameters for the constants used in the system.
Parameters_DH.m	Parameters used for Denavit Hartenberg convention
controllervel.m	Setup for NLMPC controller using fmincon.
plotRobot.m	Plotting the animation for the robot following the desired trajectory.
geometricJacobian.m	Calculation for geometric Jacobian
skew.m	takes column vector u of length 3 and returns a matrix S such that $S_v = u \times v$.
drawFrame.m	Draws a frame for plotting
mstraj.m	Generates a smooth trajectory with blends

21. Simulating in MATLAB

Setting up a run script

Several scripts were made to simulate the system with two different controllers. All the simulations ran at a time step of 10 milliseconds. This time step was used to increase the resolution of the trajectory.

Ordinary differential equation solvers were used for the simulation of the model. To save time, ODE4.m was used for the non-linear model to reduce the simulation time. The drawback was the reduced ac-

curacy. However, for the number of iterations the optimization run, the 4 th order Runga Kutta solver was enough to provide good results.
ODE45.m was used to simulate the model for LQR control. ODE45 is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. It is a single-step solver in computing $y(t_n)$, it needs only the solution at the immediately preceding time point, $y(t_{(n-1)})$ **[19]**.

Animating a script

The code used to animate the robot in Matlab is shown in Appendix B.
The code utilised *plot3*, a Matlab inbuilt function that allows you to plot 3 dimensional vectors. A loop was made based on the duration of the simulation which called the function at each time step to draw a frame of the beam type 3D printer, thus providing an animated effect.

21.1. Trajectory Generator Applied to Controlled System

We can apply the trajectory generator to produce a path for the LQR crane system to follow. The University logo is an excellent example, as it has both sharp edges and smooth curves.

Figure 14: University of Newcastle logo

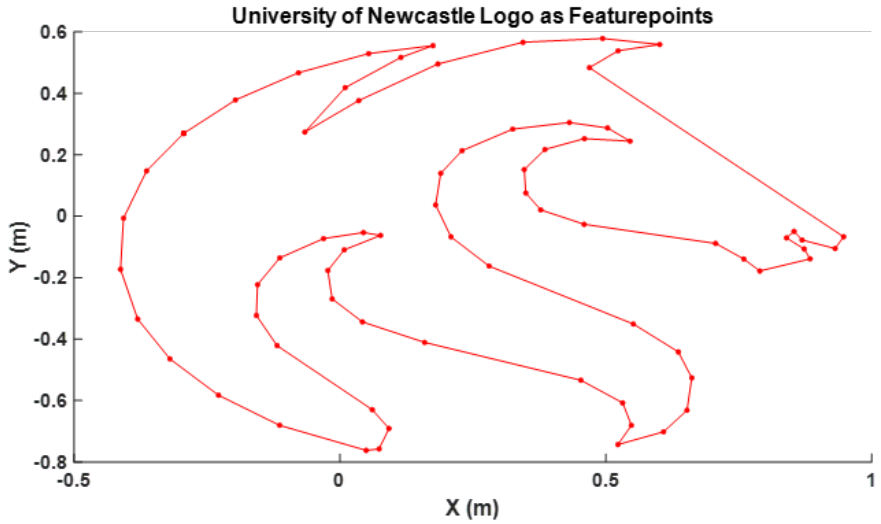

Figure 15: Feature points extracted from The University of Newcastle's logo

Figure 15 above shows the feature points taken from The University of Newcastle's logo. The sharp corners and smooth curves can be seen. These points will be used for trajectory generation to produce the desired trajectory.

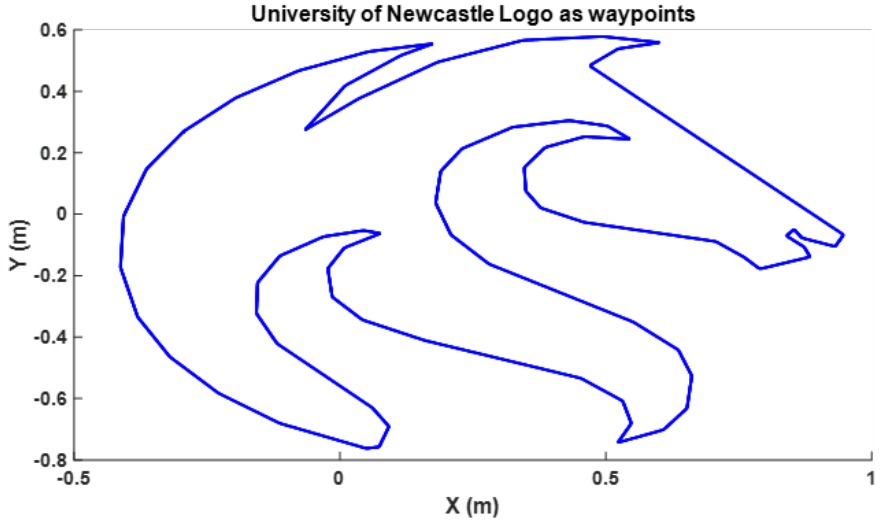

Figure 16: Waypoints extracted from logo

The desired trajectory is generated using the feature points provided. The points are spaced at 10 milliseconds. Please note that what appears to be a blue line are 3,806 points packed together.

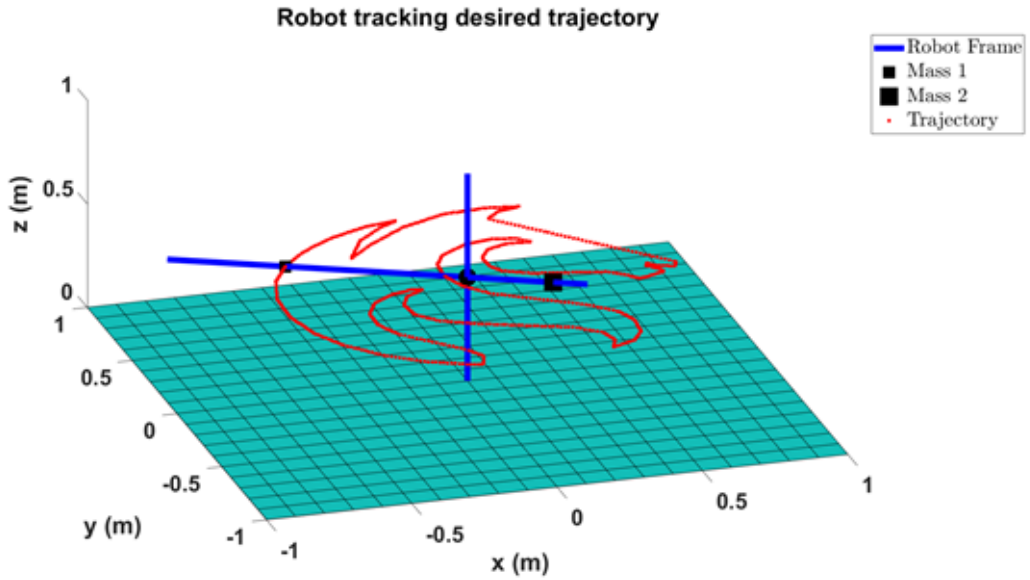

Figure 17: The simulated tower crane model tracking the trajectory

Using the forward kinematic model, figure 17 was created using plot3 function in Matlab to obtain a simulated visualization of the crane system operating. The visualization simulation was done for every 10th step in the figure above. This is really helpful to diagnose if the system is not working as expected.

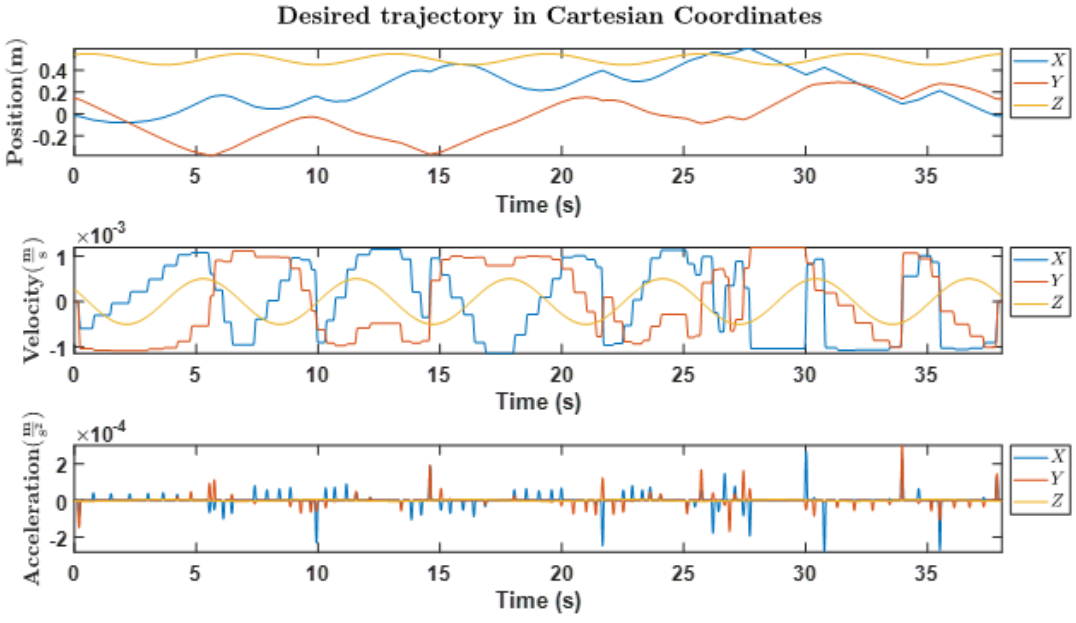

Figure 18: The desired trajectory in task space coordinates

The trajectory then transforms through inverse kinematics from the cartesian space to the joint

space

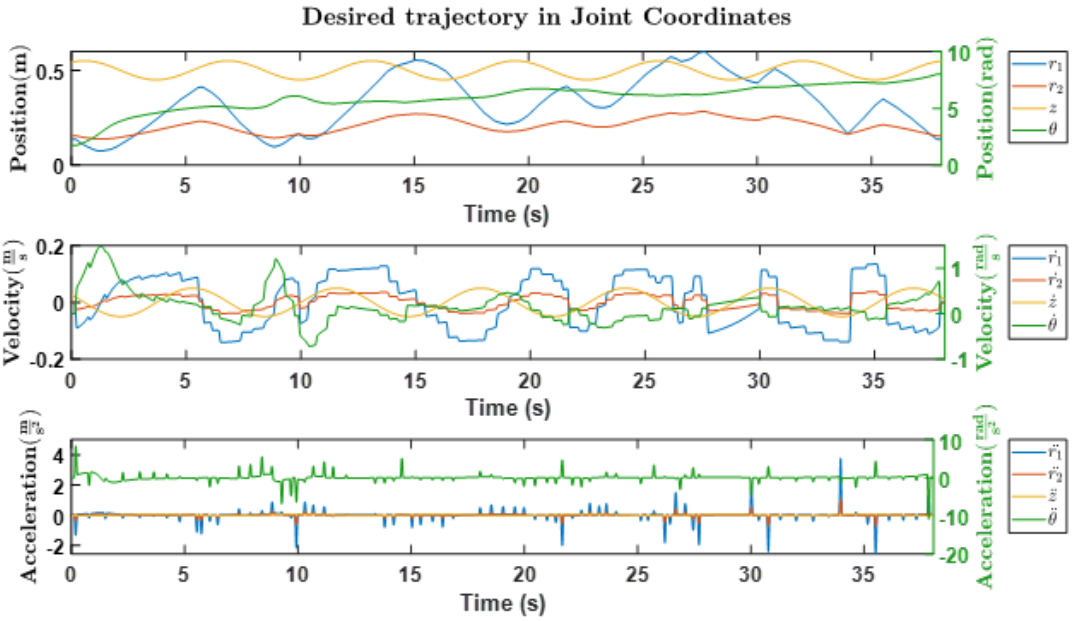

Figure 19: The desired trajectory in joint space coordinates

22. Using LQR

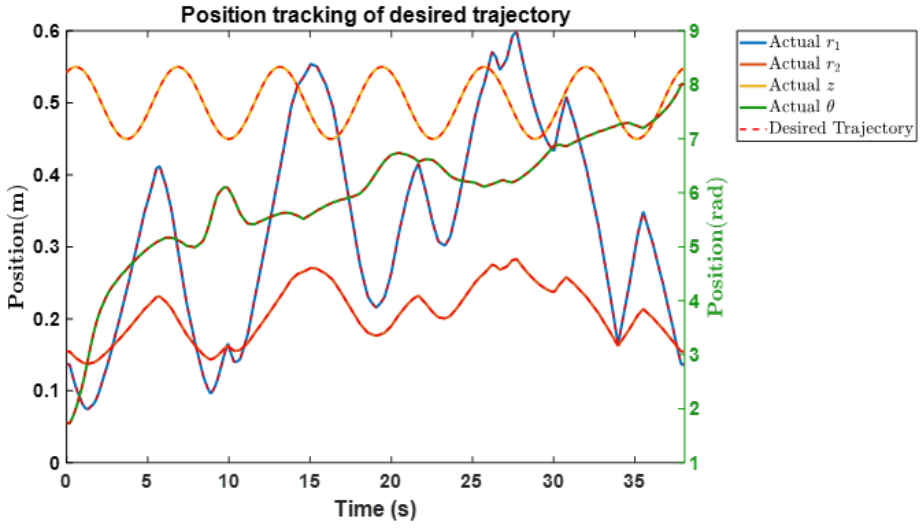

Figure 20: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

Fig. 16:The desired trajectory in joint space coordinates

Is this just desired? Or actual? Or both, whats happening in this image?

23. Using LQR

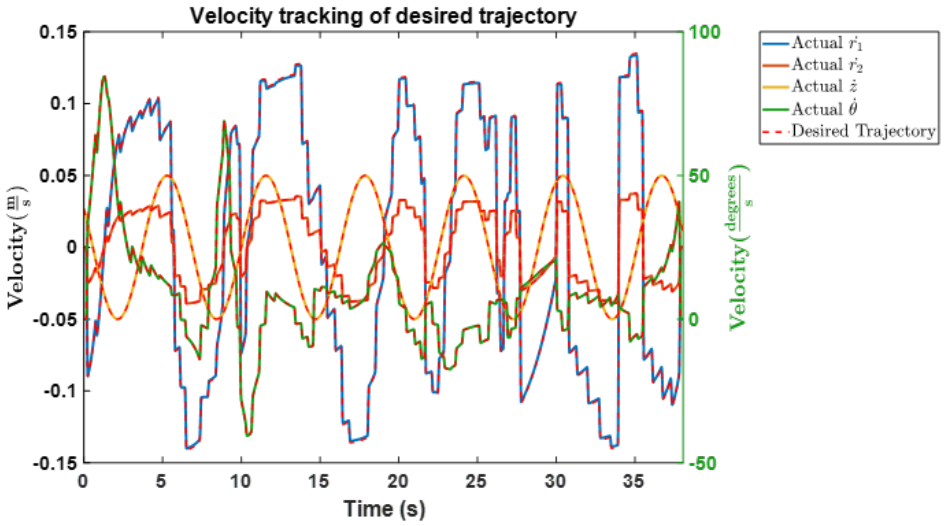

Figure 21: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

Fig. 17:Simulated results of the LQR controller tracking a desired trajectory

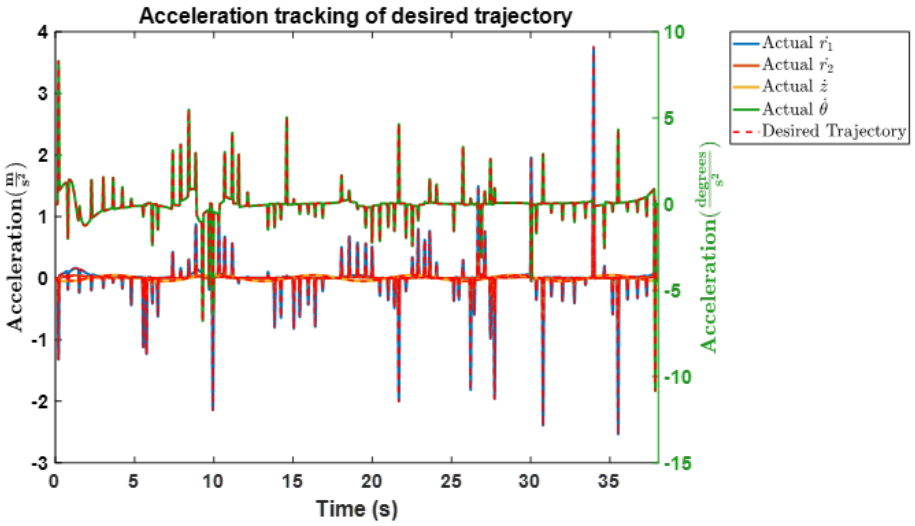

Figure 22: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

Fig. 18:Simulated results of the LQR controller tracking a desired trajectory

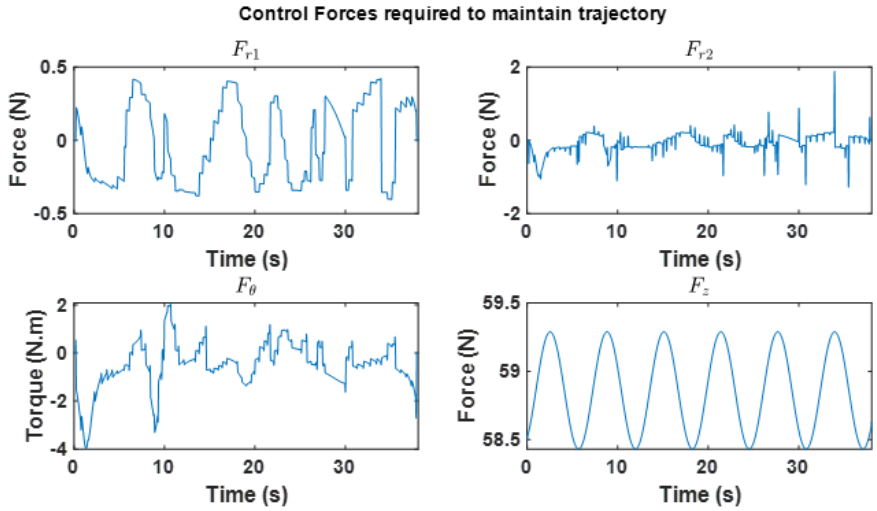

Figure 23: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

Fig. 19:Acceleration tracking

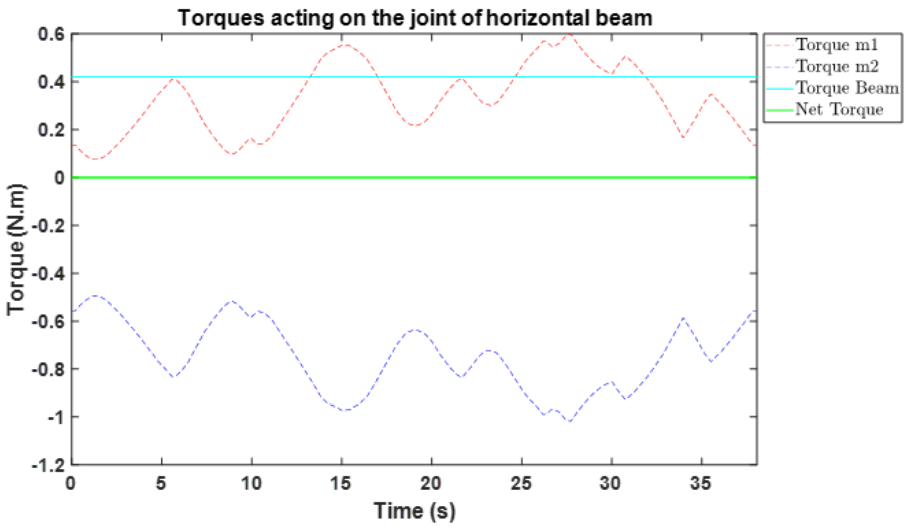

Figure 24: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

Fig. 20: Control forces applied by actuators

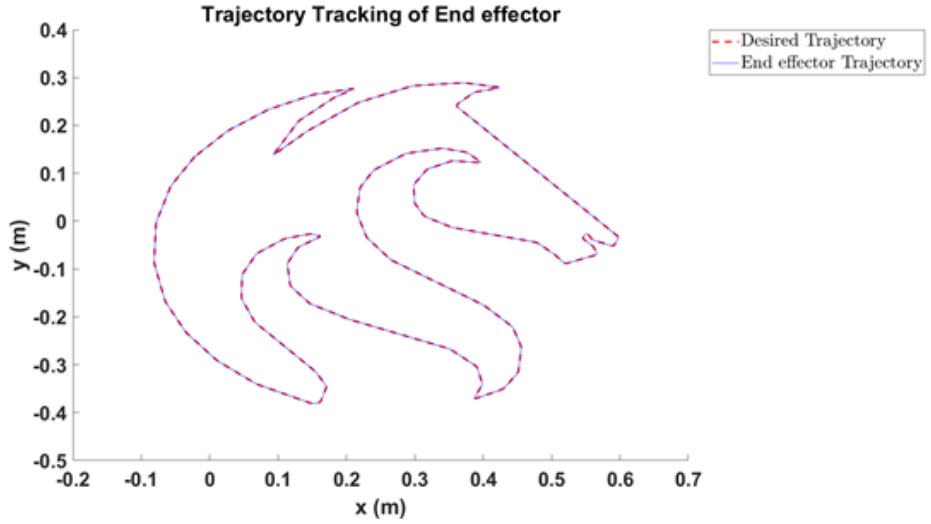

Figure 25: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

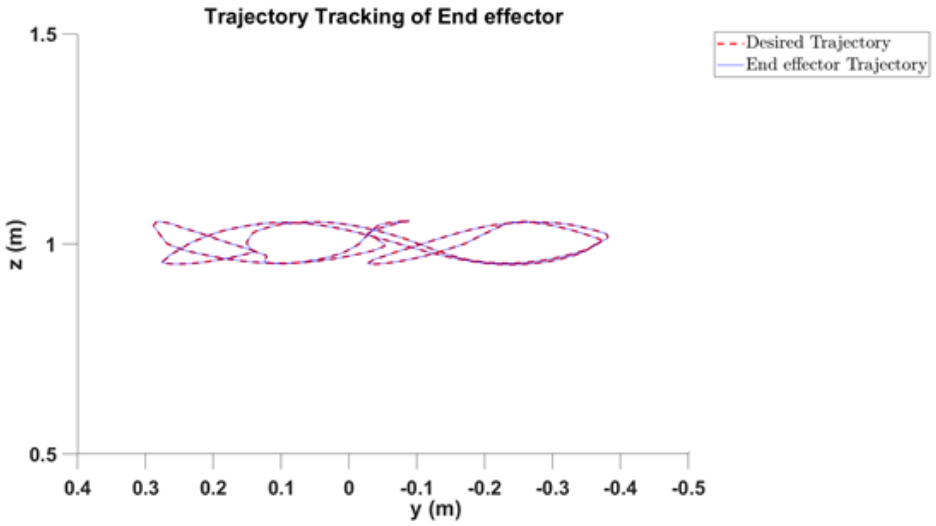

Figure 26: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

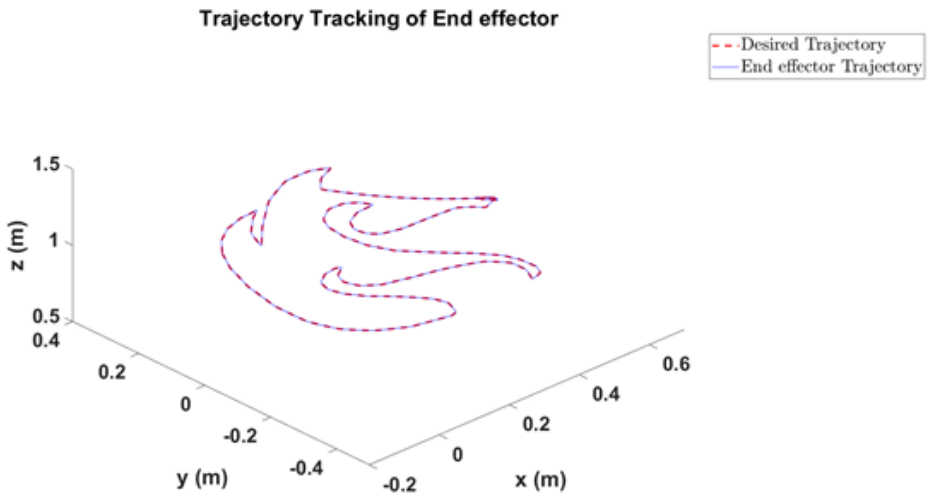

Figure 27: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

Fig. 21:Sum of Moments acting on the vertical beam Note how the sharp the edges of the seahorse is

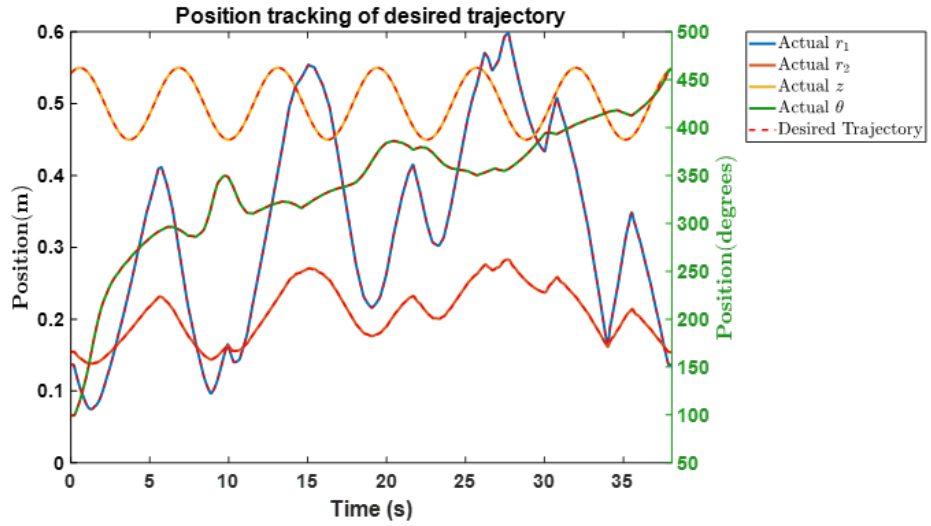

Figure 28: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

Fig. 22:Desired trajectory overlaid with end-effector trajectory Simulation of robot end effector following desired trajected

24. Using NLMPC

MPC FORMULATION [TODO] MPC FORMULATION MPC RESULTS

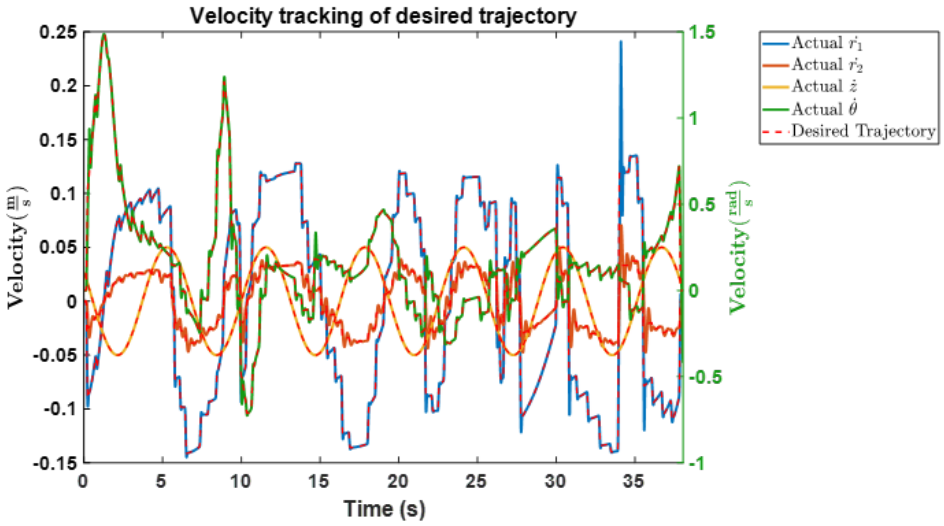

Figure 29: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

Fig. 23:position tracking using MPC

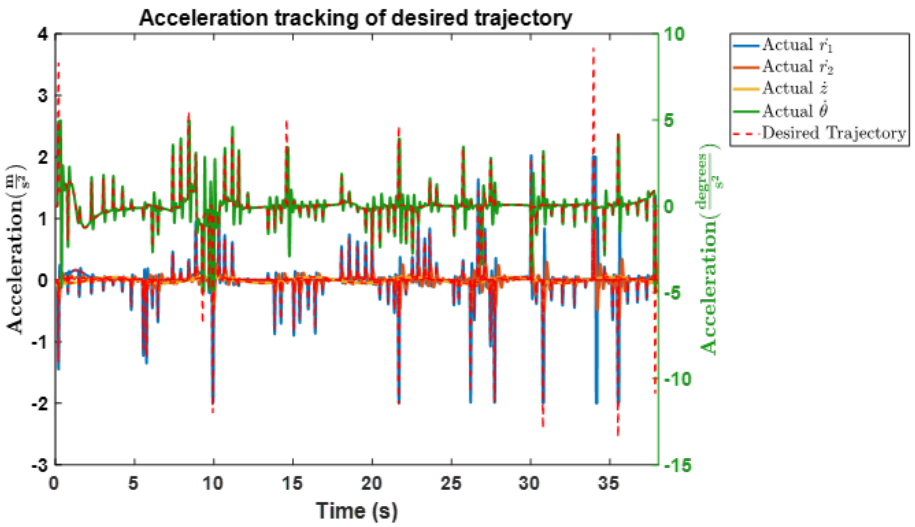

Figure 30: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

Fig. 24:velocity tracking using MPC

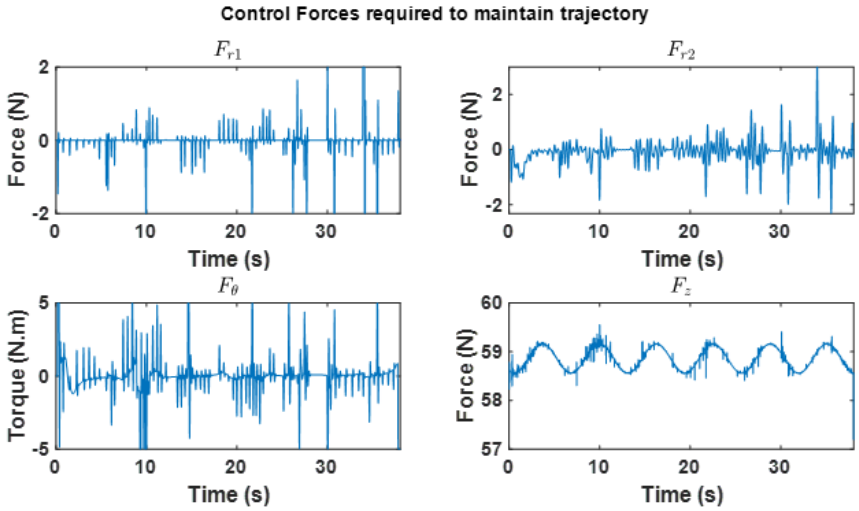

Figure 31: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

Fig. 25 Acceleration tracking using MPC

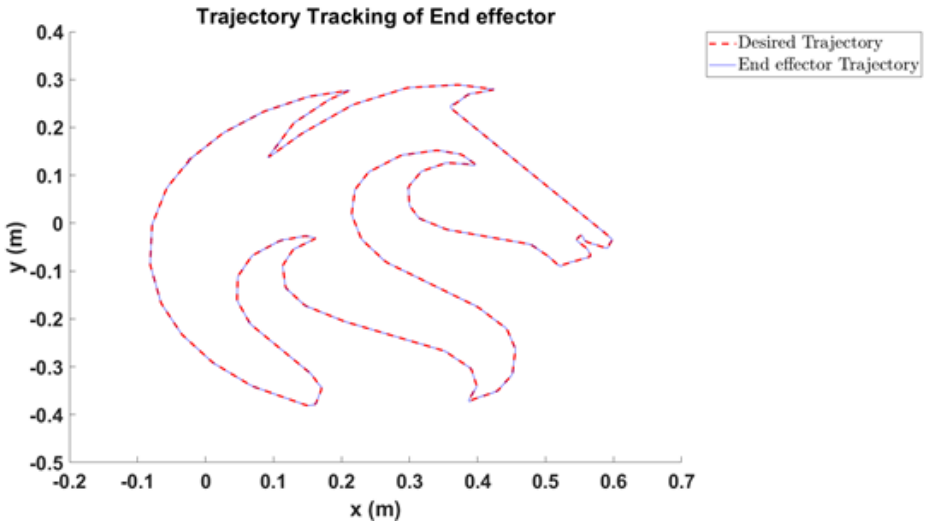

Figure 32: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

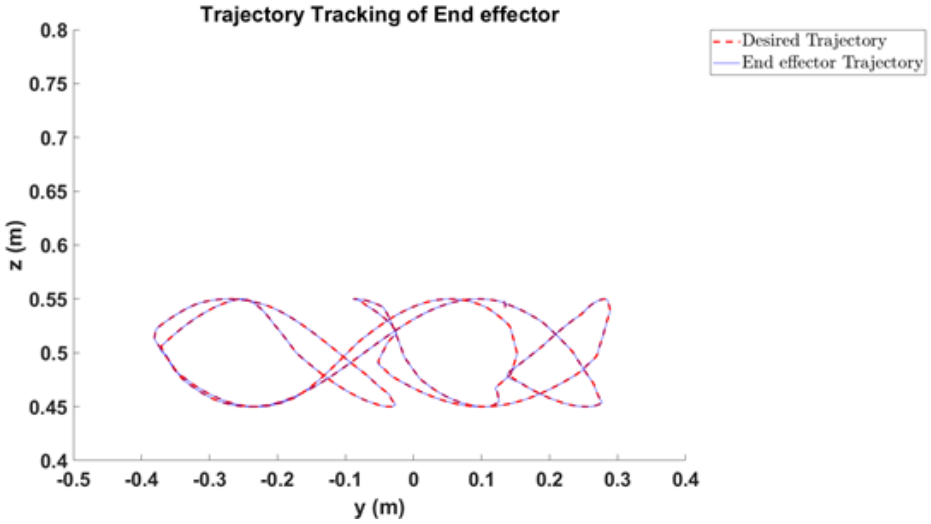

Figure 33: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

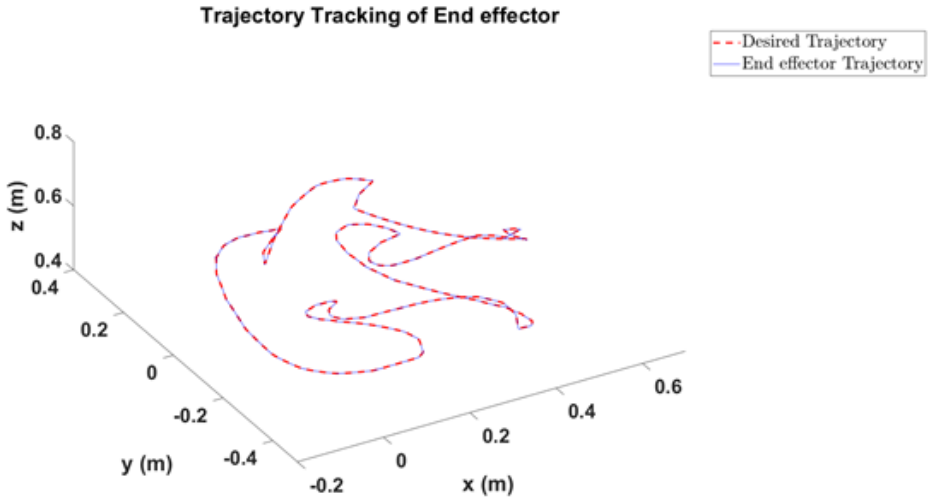

Figure 34: A common 3 axis cartesian FDM 3D printer commonly consisting of 3 actuators

Fig. 26:Control forces demanded by MPC

25. Discussion

25.1. LQR

Did it work? What do the resulting graphs say?
How difficult was it to implement?
Was the speed of computation useful for tuning the system?
Is this an ideal system for a real system?
How robust is it? Is it better than PID control? (electricity costs money and this is optimal)

Overall, what have you learned about the use of tragectory generators with respect to LQR

25.2. NLMPC

Did it work? What do the resulting graphs say?
How difficult was it to implement?
Was the speed of computation useful for tuning the system?
Is this an ideal system for a real system?
How robust is it? Is it better than LQR control? (electricity costs money and this is optimal)
Was it hard to implement trajectory generation? Why? Why not?

25.3. Comparison

Chopping and changing MPC is hard

26. Conclusion

This is one of the most important parts of the report. In the conclusion section, you should
briefly summarise the results,
"In this report, a crane model was developed using lagrangian mechanics. Two suitable modern control methods were developed and applied to the system in a coordinated simulation. thing here) was used.
reflect on the work presented,
Overall, the system was successful and could be applied to real systems and im going to explain why
make recommendations
Which control method would you pick, which trajectory method do you suggest, and explain why
suggest future work or improvements.

If someone were to follow your work, what are the next three steps they would have to take in order to actually make money off this project
Be careful with feedback linearization
Use robust feedback linearization
Make sure observability, in real life you need a nonlinear observer

References

A. Appendix A

Table 3: Definition of symbols

Physical Characteristics	**Symbology**	**Units**
Acceleration of gravity	g	m/s^2
Distance of nozzle from centre of the beam	r_1	m
Distance of counterweight from centre of the beam	r_2	m
Mass of nozzle	m_1	kg
Mass of Counterweight	m_2	kg
Angular position of the tower	θ	rad
Generalized coordinates	q	m, rad
Generalized velocities	$\dot{q}$	$m/s, m/rad$
Kinetic Energy	K	J
Potential Energy	U	J
Lagrangian	L	J

B. Appendix B

Listing 1: Matlab robot animation code.

```
function [] = plotRobot(pose,flag)
Parameters_DH;
r1 = pose(1);
r2 = pose(2);
q  = pose(3) - pi/2;
z  = pose(4);
% Denavit Hartenberg Matrix
T = @(q,d,a,alpha) [cos(q) -sin(q)*cos(alpha) sin(q)*sin(alpha) a*cos(q) ;
                    sin(q) cos(q)*cos(alpha) -cos(q)*sin(alpha) a*sin(q);
                      0 sin(alpha) cos(alpha) d;
                      0 0 0 1];
%% calculate transofrmation matricies
A01 = T(q,m1_d1,m1_a1,alpha_m1_1); % same for all masses
% Transformation Matrices for m1
A12_m1 = T(0,z,m1_a2,alpha_m1_2);
A23_m1 = T(0,r1,m1_a3,alpha_m1_3);
% Transformation Matrices for m2
A12_m2 = T(0,z,m2_a2,alpha_m2_2)*[eul2rotm([0,0,pi],'XYZ') [0;0;0]; 0 0 0 1];
A23_m2 = T(0,r2,m2_a3,alpha_m2_3);
% Transformation from Ground to each joint also 3D matrices
T01 = A01;
T02_m1 = A01*A12_m1;
T03 = A01*A12_m1*A23_m1;
T04 = A01*A12_m2*A23_m2;
% Just to plot a fixed beam
T0_fixedbeam_m1 = A01*A12_m1*T(0,lb_m1,m1_a3,alpha_m1_3);
T0_fixedbeam_m2 = A01*A12_m2*T(0,lb_m2,m1_a3,alpha_m1_3);
%collecting little r's from Transformation matrices
rB_mc1 = T02_m1(1:3,4);
rB_m1= T03(1:3,4);
rB_m2= T04(1:3,4);
rB_beam_m1 = T0_fixedbeam_m1(1:3,4);
rB_beam_m2 = T0_fixedbeam_m2(1:3,4);
```

```
%% Plotting
figure(5)
fig(1) = plot3([0,0],[0,0], [0, 1], 'b-','LineWidth',6,...
    'MarkerSize',4,'MarkerFaceColor',[0 0 0],'MarkerEdgeColor',[0 0 0]);
hold on
axis([-1 1 -1 1 0 1]);
view(0,90)
title('Robot tracking desired trajectory')
xlabel('x','Fontsize',20);
ylabel('y','Fontsize',20);
zlabel('z','Fontsize',20);
set(gcf,'color','w');
fig(2) = plot3([rB_mc1(1), rB_beam_m1(1)],[rB_mc1(2), rB_beam_m1(2)], ...
    [rB_mc1(3), rB_beam_m1(3)],'b-','LineWidth',6,'MarkerSize',4,...
    'MarkerFaceColor',[0 0 0],'MarkerEdgeColor',[0 0 0]);
fig(3) = plot3([rB_mc1(1), rB_beam_m2(1)],[rB_mc1(2), rB_beam_m2(2)],...
    [rB_mc1(3), rB_beam_m2(3)],'b-','LineWidth',6,'MarkerSize',4,...
    'MarkerFaceColor',[0 0 0],'MarkerEdgeColor',[0 0 0]);
fig(4) = plot3([rB_m1(1)],[rB_m1(2)], [rB_m1(3)], 'rs','LineWidth',6,...
    'MarkerSize',4*2,'MarkerFaceColor','k','MarkerEdgeColor','k');
fig(5) = plot3([rB_m2(1)],[rB_m2(2)], [rB_m2(3)], 'rs','LineWidth',6,...
    'MarkerSize',4*4,'MarkerFaceColor','k','MarkerEdgeColor','k');
fig(6) = plot3(0,0, rB_m2(3), 'ro','LineWidth',6,'MarkerSize',4*3,...
    'MarkerFaceColor','k','MarkerEdgeColor','k');
fig(10) = plot3([rB_m1(1)],[rB_m1(2)], [rB_m1(3)], 'rs','LineWidth',2,...
    'MarkerSize',2,'MarkerFaceColor','r','MarkerEdgeColor','r');
[x1 , y1] = meshgrid(-1:0.1:1); % Generate x and y data
z1 = zeros(size(x1, 1)); % Generate z data
surf(x1, y1, z1) % Plot the surface
drawnow
if flag
    set(gcf,'color','w')
    set(gca,'fontweight','bold','fontsize',22)
    legend(fig([1,4,5,10]),{'Robot Frame','Mass 1','Mass 2',...
        'Trajectory'},'interpreter','latex','Location','northeastoutside')
    xlabel('x (m)')
    ylabel('y (m)')
    zlabel('z (m)')
else
delete([fig(1),fig(2),fig(3),fig(4),fig(5),fig(6)])
end
end
```