6.8210 - Multi-Quadrotor Slung Load Carrying in Drake

Harvey Merton, Kevin Becker, and Ryan Poon

***Abstract*—Using multiple quadrotor systems to carry a slung load has seen a resurgence of interest in recent years. This paper discusses an implementation of such a system in PyDrake. An infinite horizon and a time-varying LQR controller are used for state regulation and trajectory stabilization of the system respectively. Differential flatness, implemented with a SNOPT mathematical program, is used to generate dynamically-feasible trajectories for the drones, from desired load trajectories. Although the controllers are able to stabilize multiple drone systems, persistent Drake errors prevented the full slung-load system from being successfully simulated. This meant that although the differential flatness module was able to generate dynamically feasible drone trajectories, they were unable to be visualized on the full drone system. Various errors and intermediate checkpoints are discussed.**

I. Introduction

Quadrotors (loosely referred to as 'drones') have been utilized for load carrying across a variety of different industries. From cameras in film and TV [1], spraying devices in agriculture [2] to urgent supply delivery in medicine [3]. Unfortunately, individual drones tend to be small and as such, can only carry payloads of limited sizes. Carrying cable-suspended payloads with multiple drones (see Figure 1) offers a way to overcome this barrier.

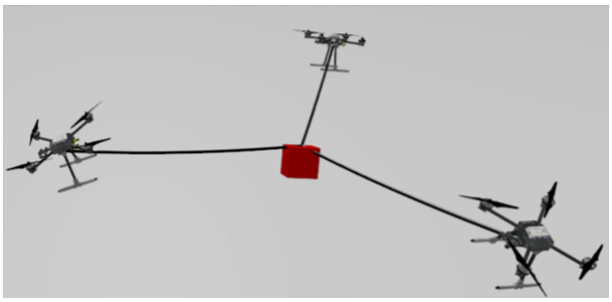

Fig. 1: Three x500 quadrotors and a slung load in a Gazebo simulation.

From 2010 to 2013, the GRASP Lab at the University of Pennsylvania published a series of papers demonstrating that multiple-drone systems can successfully carry a cable-suspended payload [4, 5, 6, 7]. The latest of these, [7], demonstrates the successful generation of trajectories that consider payload dynamics (with non-negative cable tensions) by modelling the system as a differentially flat hybrid system. They show that their method results in a load tracking error 300-400% lower than that generated by the quasi-static models that proceeded in [4, 5]. Although they discuss hybrid system dynamics for slack cables, they do not present results that demonstrate trajectories containing cables that go slack. Similarly, they present multiple formulations of possible differentially flat outputs, but only present the results for one case. Reference [8] builds on this work by using the load position and orientation as feedback to a geometric controller.

In recent years, there has been a resurgence of interest in quadrotor slung-load carrying [9, 10, 11, 12], driven by the maturity of open-source flight control software such as PX4 and Ardupilot [13]. One of the most promising areas of application is agriculture, where heavy loads need to be carried in wide-open, GPS-enabled environments.

A. Project Goals

The goal of this project is to have a functioning Drake simulation that can recreate the trajectory generation results of [7]. This includes extensive Drake and SDF modelling to get a multi-drone slung load system to function as expected. It also includes a differential flatness module that can take in a desired load trajectory and return dynamically feasible drone trajectories. Similar to the results of [7], no hybrid trajectory optimization is to be performed and so the cables remain taut throughout the selected trajectory.

We wish to extend [7]'s work by designing a time-varying LQR controller for trajectory stabilization. This will be a simpler alternative to [8]'s more advanced geometric control, and is implemented in a way to allow load position tracking in feedback (something [7] notes for future work). We begin by designing an infinite-horizon LQR controller for state regulation/stabilization at a fixed point.

Scope originally included testing multiple different deferentially flat inputs subject to wind gust disturbances. However, due to the difficulties involved in formulating the differential flatness solver as a mathematical program, and modelling the multi-drone slung load SDF in a way that worked with Drake, insufficient time remained to perform these tests.

B. Project Breakdown and Adjacent Work

Fig 2 shows how an infinite horizon and a finite horizon LQR controller were used to achieve state regulation and trajectory stabilization respectively. This figure also shows three clear sections of the project: differential flatness, trajectory stabilization and state regulation.

The project was initially divided such that Kevin would primarily work on the differential flatness section while Ryan worked on state regulation and Harvey worked on trajectory stabilization. Kevin remained the primary contributor to the

differential flatness unit. In contrast, due to the large overlap between the state regulation and trajectory stabilization implementations, most of Ryan and Harvey's work was performed collaboratively. Ryan ended up leading most of the trajectory stabilization implementation, tuning of the Q and R matrices and experiments with single drones, while Harvey lead building and debugging Drake diagrams, SDF models and multi-drone stabilizable fixed point calculations.

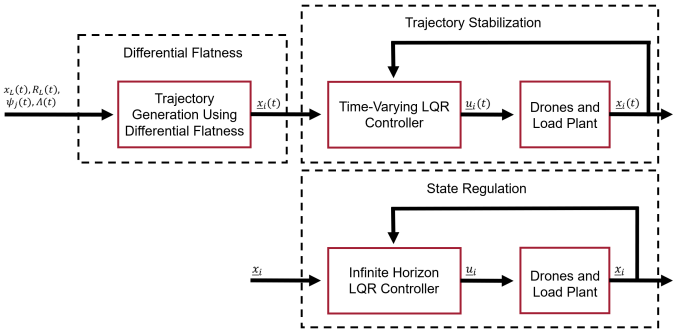

Fig. 2: Block diagrams illustrating the main components of the project and how they fit together.

This project is designed as an offshoot of Harvey's research on carrying a methane-emissions measurement device with a multi-drone system. Prior to beginning this project, we had successfully set up a basic Gazebo simulation of a three-drone and slung load system (Figure 1) modelled as an SDF. No custom flight controller algorithms had been implemented however; PX4's inbuilt cascaded PID controllers were being used.

All work performed on developing LQR controllers for stabilization and trajectory generation, as well as differential flatness were developed exclusively in Drake, for this project. They will have to be adapted for use in the main research project and would not have been investigated if not for the project. Even the original multi-drone slung load SDF file from the research project had to be modified beyond recognition to simplify the tethers, make the load function as a rigid body, alter propeller modelling and allow a stabilizable fixed point to be found in the multi-drone case.

The remainder of this paper first gives an overview of how the multi-drone slung load system dynamics, and the differential flatness modules, were implemented. The results section then discusses what the state regulation and trajectory stabilization controllers can achieve, issues with implementation in Drake and attempts to overcome these issues. Drone trajectories produced by the differential flatness module are also presented.

II. Method

A. Dynamics modelling

A complete dynamics model for the multi-drone rigid-body slung load system is developed in [7]. This model, based on the Euler-Newton method, is presented here. Fig. 3 illustrates the specific case when three drones are used, which is the case developed throughout the rest of this paper. The system

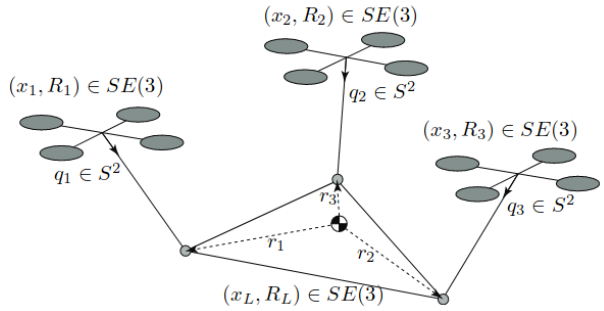

Fig. 3: Three quadrotors carrying a rigid body slung load [7].

of equations 1 completely describes the motion of this system, where the symbols used are defined in Table I. The symbols are selected to be consistent with those used in [7].

$$m_i \ddot{x}_i = f R_i e_3 - m_i g e_3 + R_L T_i q_i \tag{1a}$$
$$J_i \dot{\Omega}_i + \Omega_i \times J_i \Omega_i = M_i \tag{1b}$$
$$m_L \ddot{x}_L = -\sum R_L T_i q_i - m_L g e_3 \tag{1c}$$
$$J_L \dot{\Omega}_L + \Omega_L \times J_L \Omega_L = \sum r_i \times -T_i q_i \tag{1d}$$

Equations 1a and 1b describe the dynamics of drone i (where $i \in \{1, 2, 3\}$ for the 3-drone case). Similarly, equations 1c and 1d describe the load dynamics. Equations 1a and 1c simply represent the sum of forces acting on the drone and load respectively, whilst equations 1b and 1d represent the sum of moments on the same.

Several assumptions (also discussed in [7]) are made in deriving this system. These include:

1) The cables are massless and constant length.
2) The cables are attached to the quadrotors at their centres of mass.
3) Aerodynamic drag acting on the quadrotors, load, and cables, is negligible.
4) The cables are always taut. The hybrid dynamics case that can handle slack cables is discussed in [7], but is not presented here.

As the SDF models capture the physics of the multi-drone slung load system, these equations do not need to be implemented to model the plant. They are however used as constraints in the mathematical program solver for the differential flatness module.

The full SDF model used for the three-drone and load system can be seen in Figure 4. It is pictured in a stabilizable fixed point calculated with static analysis ($\sum F = \sum M = 0$) like all linear and angular velocities at a fixed point). The model features three tethers each modelled as single rigid cylindrical rods attached with ball joints at either end. Each tether is generated from a XACRO file to allow the length, mass and number of links to be easily changed. The load is modelled as a sphere with the tether attachment points symmetrically distributed along the center plane to simulate the rigid body aspect of the load. The tethers are connected to the drones at their centers of mass.

$m_L \in \mathbb{R}$	Mass of load.
$J_L \in \mathbb{R}^{3\times3}$	Inertia matrix of the load with respect to the body-fixed frame.
$R_L \in SO(3)$	The rotation matrix of the load from the body-fixed frame to the inertial frame.
$\theta_L \in \mathbb{R}^3$	Orientation of the load as roll-pitch-yaw (RPY) angles in the inertial frame. This is R_L converted to RPY Euler angles.
$\Omega_L \in \mathbb{R}^3$	Angular velocity of the load in the body-fixed frame.
$x_L \in \mathbb{R}^3$	Position vector of the center of mass of the load in the inertial frame.
$m_i \in \mathbb{R}$	Mass of ith quadrotor.
$J_i \in \mathbb{R}^{3\times3}$	Inertia matrix of the i^{th} quadrotor with respect to the body-fixed frame.
$R_i \in SO(3)$	The rotation matrix of the i^{th} quadrotor from the body-fixed frame to the inertial frame.
$\Omega_i \in \mathbb{R}^3$	Angular velocity of the i^{th} quadrotor in the body-fixed frame.
$x_i \in \mathbb{R}^3$	Position vector of the center of mass of the i^{th} quadrotor in the inertial frame.
$f_i \in \mathbb{R}$	Thrust produced by the i^{th} quadrotor.
$M_i \in \mathbb{R}^3$	Moment produced by the i^{th} quadrotor.
$\psi_i \in \mathbb{R}$	Yaw angle of the i^{th} quadrotor.
$q_i \in \mathbb{S}^2$	Unit vector from the i^{th} quadrotor to its attachment point on the load in body-fixed frame of the load.
$L_i \in \mathbb{R}$	Length of the cable between the i^{th} quadrotor and the load.
$r_i \in \mathbb{R}^3$	Vector from the center of mass of the load to the attachment point of the ith quadrotor to the load.
$T_i \in \mathbb{R}$	Tension in the the cable between the i^{th} quadrotor and the load.
$e_1, e_2, e_3 \in \mathbb{R}$	Standard unit vector along x, y, z axes in the world frame.

Table I: VARIOUS SYMBOLS USED THROUGHOUT.

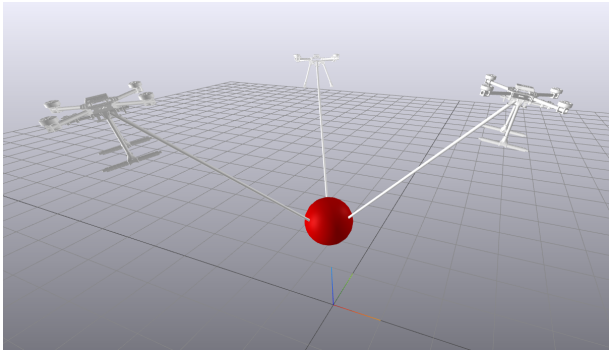

Fig. 4: Meshcat screenshot of the load with three tethers and three drones at a stabilizable fixed point.

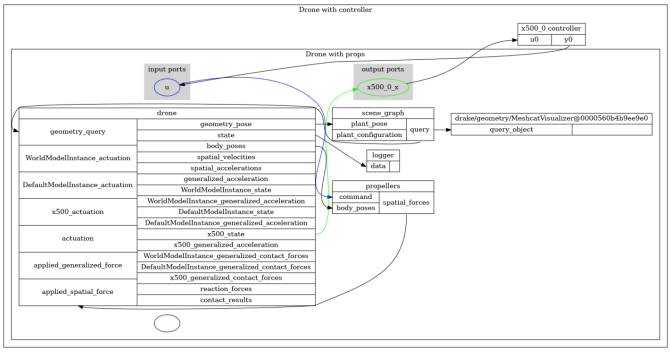

Fig. 5: Block diagram of a single drone plant with an infinite horizon LQR controller. It should be noted that this block diagram looks the same for a single drone with a finite horizon LQR controller.

Each drone SDF represents a modified version of the Holybro x500 drone which is being used in the associated research project. The main modification performed was removing the propellers as propeller inputs are modelled with Drake's propeller class. It took much debugging to find that a stabilizable fixed point could not be found during controller design without removing the propeller as they had mass and inertia that moved the drone's overall center of mass. Four inputs are available to each drone and represent the propeller speed (and so torque and thrust) at each drone's propeller locations.

B. Simulation Method

1) Single Drone Case: Implementing the infinite horizon controller for the single drone case was relatively simple. Using the quadrotor example from the Underactuated Robotics website as a template, the SDF file for a Hollybro X500 drone was first uploaded. The plant contains a floating roll-pitch-yaw joint to avoid having to use quaternions to define the state, and the propeller objects were connected to the spatial forces input port. Four propellers acted as the input u, and the state x was the output of the plant as seen in Figure 5.

The LQR controller for the plant linearized about a user-defined state is then created and connected to the plant. The Q matrix was set to be a diagonal matrix with the first half of the diagonal elements (corresponding to the position variables) set to 10 and the other half (corresponding to the velocity variables) set to 1. The R matrix was simply set to a diagonal matrix of 0.1. A block diagram of the fully connected plant and controller can be seen in Figure 5.

Moving to a finite horizon, time varying LQR controller posed some challenges. Before implementing the finite horizon LQR controller, state and input trajectory objects were first created using direct collocation between the initial state and the desired final state. Costs were enforcesd on the control effort as well as the time duration of the entire trajectory. These objects were set as options for the Drake function that generates the finite horizon LQR controller, and the Q and R matrices were set as the same as in the single drone infinite horizon LQR controller case. Once the controller and plant were connected, the block diagram resembles the same as the infinite horizon LQR controller case in Figure 5.

To ensure that the controller switches to an infinite horizon LQR controller at the end of the trajectory, the plant was linearized about the final state of the trajectory, and the S matrix solved for in the algebraic Riccati equation was

extracted from the appropriate Drake functions. The final cost matrix Q_f was set to equal S in order to activate the infinite horizon LQR controller at the end of the trajectory.

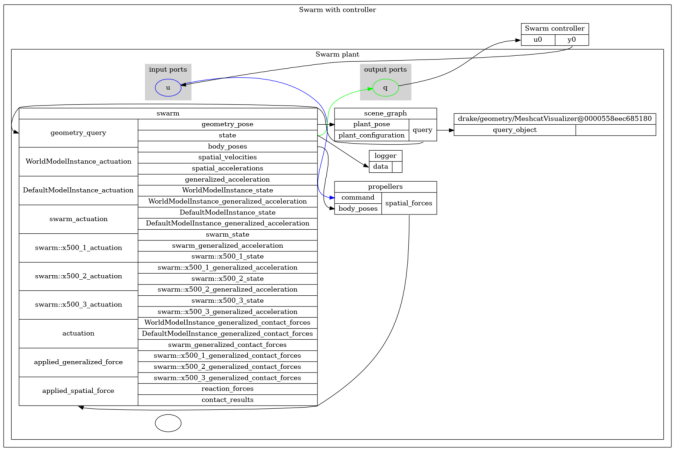

Fig. 6: Block diagram of three drone plant with a controller. While the propeller block looks the same as Figure 5, note the extra actuation sections under the "swarm" plant. It should be also pointed out that this block diagram looks the same for three drones regardless of whether the LQR controller is infinite horizon or finite horizon.

2) Three Drone Case: Originally, the plant involved a multiplexer and demultiplexer to concatenate the inputs and outputs of the plant. This caused some issues, due to an unintentional mix up of the state variables. In an attempt to debug the issue, the multiplexer and demultiplexer were removed, instead all 12 inputs were funneled into the plant and all 36 state variables out of the plant. When the issue was still not resolved, it was realised that the state ordering had all drone positions before all drone velocities (each drone's state could not simply be concatenated). Using the full system state as output solved this issue. A block diagram of the system can be seen in Figure 6. Once again, a roll-pitch-yaw joint was added to each quadrotor, and the LQR controller was designed around stabilizable state of size 36. The diagonal Q and R matrix were triple the length of those in the single drone case, but their composition was still the same.

Creating a finite horizon LQR controller for three drones, especially with the script written for the infinite horizon LQR controller, was fairly trivial. In the same manner as the process described in Section II-B1, the Q matrix, R matrix, and stabilizable state was adjusted accordingly to take into account the extra drones. The only major change in simulation quality was the solver time, which was noticeably longer.

3) Adding Tethers and Load Method 1: Adding the load and the tethers into the simulation posed a very difficult challenge that has yet to be solved. First, the kinematic tree was arranged via the SDF file such that the parent link is the load, followed by the tethers and drones at child links (Figure 7). The tether, originally made of several links, was simplified to be a single link with ball joints at the ends. Because three roll-pitch-yaw

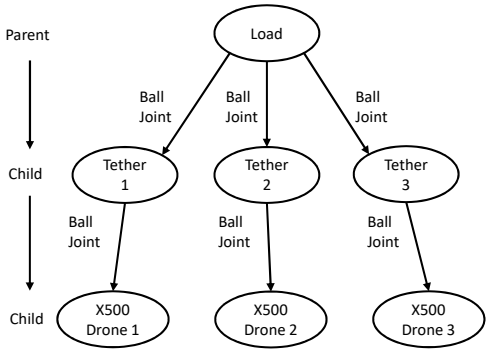

Fig. 7: Kinematic tree of the three drones, tethers, and load case. Note the parent link in this case is the load, followed by the three tethers and then the drones.

joints could not be attached to the three drones in the same multibody, the roll-pitch-yaw joint was attached to the load rather than the drones.

Because Drake automatically adds the degrees of freedom for each new model to the state vector of the system, full state feedback LQR on the entire system had to take into consideration not just the drone poses but also the load and tether poses. In reality, our differential flatness algorithm would provide trajectories for the three drones, and ideally a physics engine could handle the poses of the tethers and the load based on the poses of the drone. However, we believed that with the statics relationships between the drones, tethers, and load, we could easy find a stabilizable state and create a function that would output the state of the load and tethers given the poses of the drones (or vice versa).

Furthermore, a simpler version of the problem was also attempted by simplifying the problem to a single drone with the tether and load. This was done to better visualize a segment of the system following a trajectory. In this case, the load is still the parent followed by the tether and drone child links. The position state variables could be defined as follows:

$$\mathbf{q}_{load} = \begin{bmatrix} x & y & z & \phi & \theta & \gamma \end{bmatrix}^T \tag{2}$$

$$\mathbf{q}_{tether} = \begin{bmatrix} \phi_{lt} & \theta_{lt} & \gamma_{lt} \end{bmatrix}^T \tag{3}$$

$$\mathbf{q}_{drone} = \begin{bmatrix} \phi_{td} & \theta_{td} & \gamma_{td} \end{bmatrix}^T \tag{4}$$

Where the first six variables represent the position coordinates of the load (x-, y-, and z- positions followed by the roll, pitch, and yaw) and ϕ, θ, and γ respectively represent the roll, pitch, and yaw of the ball joints. The subscript lt represents the joint connecting the load to the tether, and td represents the joint between the tether and the drone.

4) Adding Tethers and Load Method 2: A second method was also attempted to achieve visualization of the three drone system following a trajectory. This time, the kinematic tree was flipped over, with the parent link being the drone and the child links being the tether and the load (Figure 8). This was

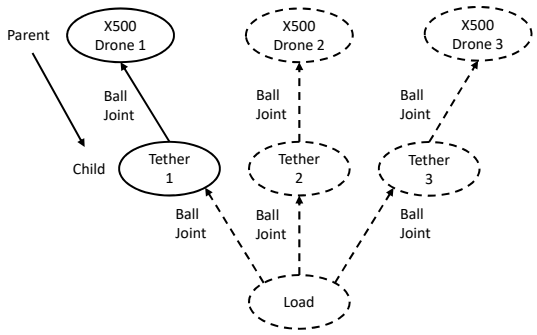

Fig. 8: The kinematic tree of the drones, tethers, and load, except with the parent starting at the drone. The solid lines (the single drone with a single tether) represents the kinematic tree with which an infinite horizon LQR controller was successfully implemented. However, when the load and two other drones were added, errors were encountered.

to take a cart-pendulum-like approach to the problem, which had better documentation online. At first, only a single drone with a tether attached via a ball joint was attempted. The state could be defined as such:

$$\mathbf{q}_{drone} = \begin{bmatrix} x & y & z & \phi & \theta & \gamma \end{bmatrix}^T \tag{5}$$

$$\mathbf{q}_{tether} = \begin{bmatrix} \phi_{dt} & \theta_{dt} & \gamma_{dt} \end{bmatrix}^T \tag{6}$$

Where the first six variables represent the position coordinates of the drone (x-, y-, and z- positions followed by the roll, pitch, and yaw) and ϕ, θ, and γ respectively represent the roll, pitch, and yaw of the ball joint between the tether and drone. The following step was to add the model of the load into the SDF file, but that was as far as was attempted.

C. Differential Flatness Modeling

Differential flatness is a property of certain systems that enables closed form solutions of trajectories that are differentiable k times. For a given system,

$$\dot{x} = f(x, u)$$

to be differentially flat, the following properties from [14] must be satisfiable:

$$z(t) = h\left(x, u, \frac{\mathrm{d}u}{\mathrm{d}t}, ..., \left(\frac{\mathrm{d}^k u}{\mathrm{d}t^k}\right)\right) \tag{7a}$$

$$x(t) = X\left(z, \frac{\mathrm{d}z}{\mathrm{d}t}, ..., \left(\frac{\mathrm{d}^k z}{\mathrm{d}t^k}\right)\right) \tag{7b}$$

$$u(t) = u\left(z, \frac{\mathrm{d}z}{\mathrm{d}t}, ..., \left(\frac{\mathrm{d}^k z}{\mathrm{d}t^k}\right)\right) \tag{7c}$$

We knew the drone with a suspended load system is differentially flat [7], so we had to choose our flat outputs, z. One flat output needed to be prescribed for each degree of actuation in this system. Therefore, we need set the trajectories of $4n$ (where n is the number of drones in the system) flat outputs to fully define the trajectory. For the selected flat outputs, we used the load position (x_L) and orientation (θ_L), which accounted for 6 degrees of actuation, which is possible for $n >= 3$ drones. Including the heading of each drone (ψ_i), accounts for another n degrees of actuation. This left us with 3 extra flat outputs to prescribe for our $n = 3$ drone case. The convention used in [7] will be maintained, and these outputs will be referred to as Λ. From [7], $\Lambda \in \mathbb{R}^{3n-6}$ satisfies Equation 8 (derived by rearranging Equations 1):

$$\vec{T} = \phi^+ W + N\Lambda \tag{8}$$

where $\vec{T}$, W are defined by Equations 9 and 10, and ϕ^+, N are the pseudoinverse and kernel respectively of 11:

$$\vec{T} = \begin{bmatrix} T_1 q_1 \\ T_2 q_2 \\ ... \\ T_n q_n \end{bmatrix} \tag{9}$$

$$W = -\begin{bmatrix} R_L^T(m_L(\ddot{x}_L + g e_3) \\ J_L\dot{\Omega}_L + \Omega_L \times J_L\Omega_L \end{bmatrix} \tag{10}$$

$$\phi = \begin{bmatrix} I & I & ... & I \\ \hat{r}_1 & \hat{r}_2 & ... & \hat{r}_n \end{bmatrix} \tag{11}$$

Thus, the flat outputs selected are the same as in [7]: $z = (x_L, \theta_L, \Lambda, \psi_j)$.

The functions behind the differential flatness were made to be as flexible as possible, attempting to enable the expansion to an n drone case for any trajectory. The only change to make would be to add definitions for the each Λ for each number of drones, n, for cases where $n >= 3$. An important constraint to note is that the load trajectory needs to be six times differentiable, and the tensions need to be differentiated four times to obtain the entire state and the inputs for this system.

To start, the trajectories generated in the differentially flat variables were made as piecewise polynomials. This enables the approximation of nearly any function as the order of the function inceases similar to a Taylor Series. The piecewise polynomial generator from the differential flatness example in textbook [14] was implemented. This polynomial generator used SNOPT to find a good polynomial fit.

Several sets of initial and final conditions were considered for the differential flatness. At first, the initial position was set, and the initial velocity and acceleration of the flat outputs were all fixed at zero. However, it was difficult for the solver to find a good polynomial fit, since the load trajectory was fixed for each timestep on the desired trajectory. For this project, the desired trajectory was set such that the load moved in a circle in X and Y. Therefore, to simplify the problem, the flat output initial conditions were not explicitly stated. Instead, the piecewise polynomial, along with the dynamics solver, was used to find the full set of (flat and not-flat) initial conditions. These initial conditions would be fed into the simulation.

D. Differential Flatness Dynamics Solver

To solve for the non-flat variables, the nonlinear numerical solver SNOPT was used. This enabled extra flexibility, as it can easily be generalized to N drones. The only change is that Λ would have to be set for each N drone case, which is much easier than reworking all of the dynamics for each different Λ we wanted to try. This method of filling in the remaining values was important because a portion of the project was dedicated towards testing out different variations of Λ to see how the resultant trajectories ended up. To maintain continuity with [7], we initally chose these outputs as the X and Y components of the tension in cable 1 and the X component of tension in cable 2.

The differential flatness solver inputs are the trajectories of all $4n = 12$ flat outputs, the number of timesteps, the number of drones, and the final time. It then found solutions at each timestep for:

- u on each of the $4n$ rotors
- The position, velocity, and acceleration of each drone
- The roll and pitch for each drone, along with their first two derivatives
- The tension vector for each cable
- Each lambda value

The following constraints at each timestep were used:

- For each drone
  - The input limits on each rotor
  - Yaw and pitch was limited to keep the drone within 45 degrees from upright in pitch and roll. This had the added benefit of keeping the force more reasonable for this use case. It also prevented gimbal lock from becoming an issue.
  - The cable length must be equal to the distance from the drone's center of mass to the anchor point on the load (forces the taut cable constraint)
  - The cable and tension vector are parallel (the mass-less cable assumption)
  - Constraints to limit yaw and its derivatives to be equal to what was set by the input (drones don't turn)
  - Sum of forces on the drones (Equation 1a)
  - Sum of moments on the drones (Equation 1b)
- The sum of forces on the load (Equation 1c)
- The sum of moments on the load (Equation 1d)
- The Λ constraints. For us, these were: $\Lambda = (T_1 q_1 \cdot e_1, T_1 q_1 \cdot e_2, T_2 q_2 \cdot e_1)^T$ (X and Y components of the tension in cable 1 and the X component of tension in cable 2

The ability to include certain costs was also added. The reasoning may be found in section III.

To speed up subsequent SNOPT solutions, a valid solution would be cached to a file using the pickle library. This solution would then be used to seed the solver's initial guess for the next run. The resulting trajectories were the extracted from the solver.

III. Results and Discussion

A. Meshcat Simulations

Several steps were taken to work towards carrying a slung load with a three-drone system. The first involved implementing an infinite horizon LQR controller and a finite horizon LQR controller on a single x500 drone. This was followed by a three-drone system without a slung load before finally adding the load to the three-drone system. The results of each stage are discussed below.

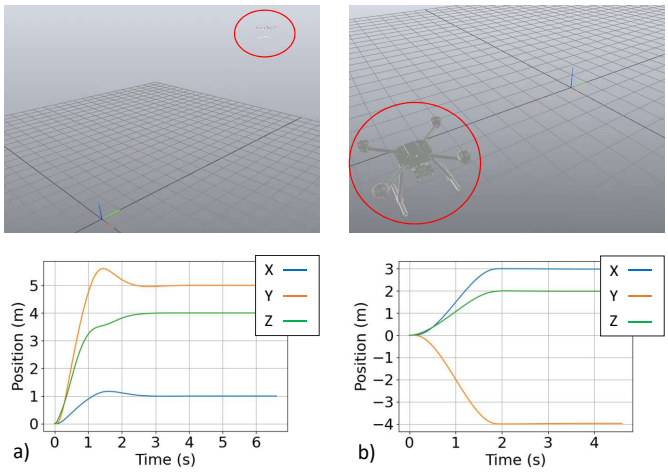

Fig. 9: Meshcat screenshot and plots of position over time for a single quadrotor using a) an infinite horizon LQR controller and b) a finite horizon LQR controller. Note that for the infinite horizon LQR controller, there is some slight overshoot before the position reaches steady state.

1) Single Drone Case: The infinite horizon LQR controller was successful for the desired stabilizable point (Figure 9a) after having been simulated via the method described in Section II-B1. However, it was noted that in most cases, the single drone overshot its steady state position quite often, especially if the stabilizable state was set to be far from the initial state. This makes sense because the single drone with the infinite horizon LQR controller isn't following a strict trajectory defined by an optimizer with constraints and costs. Furthermore, it was also noted that while the single drone eventually reached its final position in a stable manner, its roll, pitch, and yaw over the course of its movement would take on impractical values. For instance, the drone might spin upside-down while flying up towards the desired final state.

When implementing the finite horizon LQR controller as in Section II-B1, the single drone could successfully follow the trajectory given by direct collocation (Figure 9b). It should be noted that in this case, the overshoot seen with the single drone infinite horizon LQR controller. This also makes sense because the time and effort costs enforced on the direct collocation solver caused the resulting trajectory to be much more "damped" in its approach to the final state.

2) Three Drone Case: After using the method described in Section II-B2, an example end result for the three drone

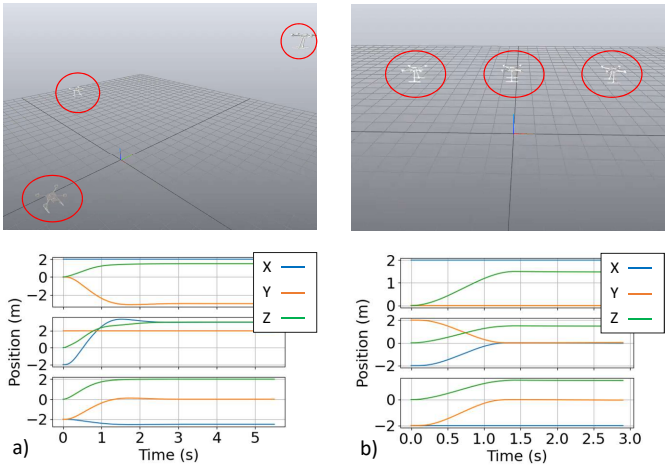

Fig. 10: Meshcat screenshot and plots of position over time for three quadrotors using a) an infinite horizon LQR controller and b) a finite horizon LQR controller. In the case of the plant with a finite horizon LQR controller, the controller switches to infinite horizon at the end of the trajectory. Note that the slight overshoots seen a) aren't seen in b).

case with an infinite horizon LQR controller can be seen in Figure 10a. Similar to the single drone case, some overshoot can be seen as the drones approach their final state, but that overshoot is gone when the finite horizon LQR controller is implemented using the procedure described in Section II-B2 (Figure 10b). These results also make sense according to the same reasoning explained in Section III-A1.

3) Integrating the Load and the Tether: Once the tethers and load were implemented using the method described in Section II-B3, a stabilizable state was able to be found and visualized in Meshcat using the dynamics equations seen in Section II (Figure 4). However, several errors were encountered attempting to get all three drones to follow a trajectory while all tethered to the load, the primary ones being that the being unable to find stable eigenvalues. As a result, while the final state could be visualized, the entire movement of the drones to that final state could not, hence the simplification of the problem to a single drone with a tether and a load. This was done while keeping the load as the parent link as described in Section II-B3. However, the error that was encountered with this simpler case was that the fixed point about which the system was linearized was not stabilizable. This could have been due to a couple of factors, such as the joint location on the load or the free rotation of the tether and the box about the axis of the tether.

To attempt to circumvent the issue, the simulation method and kinematic tree described in Section II-B4 was employed. For this case, the single drone with a tether was successfully able to be connected to an infinite horizon LQR controller linearized about a given state (Figure 11). The tether, represented as a pendulum-like single linkage with a ball joint, could even be balanced at its upright unstable fixed point seen in Figure 11b.

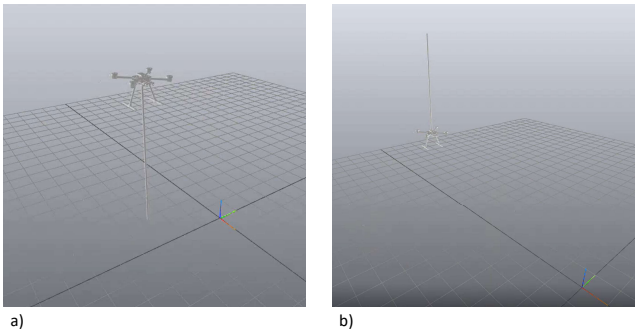

Fig. 11: Meshcat screenshots of a single drone and tether with an infinite horizon LQR controller, where a) shows the system stabilized about a point with the tether suspended underneath the drone and b) shows the system stabilized about a state where the tether is in its upright unstable position

Nevertheless, when the load was attached to the end of the tether, several problems were encountered, namely the "spatial force is not a number" and the linearized state not able to be stabilizable. It was also realized that adding two extra drones would force the states to be defined using quaternions because Drake does not allow the attachment of multiply roll-pitch-yaw joints to a single multibody system. As a result, it was deemed that the best approach to this problem of simulation visualization would best be done via the kinematic tree described in Figures 7 and 4.

B. Trajectories from Differential Flatness Algorithm

With the finite horizon controller and trajectory optimizer in place for three drones without the tether and the load, we could then move on to testing the trajectories generated by the differential flatness algorithm. The differential algorithm was seeded with the values shown in Table II, and the trajectory was generated to show what the trajectory would look like for the three drone case attempting to move the load in a full circle with a radius of 1 m over 5 seconds. This trajectory was chosen because both circles, and constant values, are infinitely differentiable.

Attempts were made to convert the trajectories from two dimensional lists to trajectories in PyDrake. Since the three drone system did not have the load, the generated trajectory was attempted to be used without the load. To reduce the impact this would have on the dynamics, the weight of the load in the solver was set to a small value. Once this trajectory with a light mass load was generated, the state was stripped to include just the drones. Then, attempts began at using the *PiecewisePolynomial.FirstOrderHold* function to convert these generated trajectories to PyDrake trajectories that the time-varying LQR can linearize about. However, our group ran out of time in the project while attempting to incorporate these two portions together.

The trajectory analyzed was a three drone system with the load moving in a flat circle perpendicular to gravity. The differentially flat output trajectory of the load may be seen in Figure 12.

Table II: VALUES USED FOR THE DIFFERENTIAL FLATNESS TRAJECTORY GENERATION, WHERE U IS THE COMMAND INPUT TO A THRUSTER. ALL DRONES WERE ASSUMED IDENTICAL.

Variable	Value	Units
Load mass	1.0	kg
Load inertia	1.07, 1.07, 1.94	$kg * m^2$
Cable length	3.0	m
Propeller thrust ratio	1.0	N/u
Propeller moment ratio	0.0245	N*m/u
Anchor point drone 0	0, 1/2, 1/2	m (x, y, z)
Anchor point drone 1	-1/2, -1/2, 1/2	m (x, y, z)
Anchor point drone 2	1/2, -1/2, 1/2	m (x, y, z)

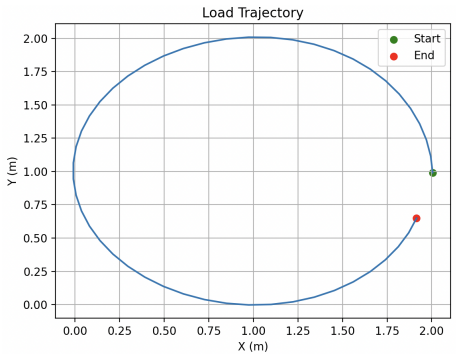

Fig. 12: Desired load trajectory output from the Piecewise Polynomial made to follow a circle. As can be seen, the Piecewise Polynomial is able to approximate a circle well.

The output trajectories for each drone may be seen in Figure 13, shown in 3 dimensions. Appendix B contains more plots showing projections of these trajectories as well as plots of several variables over time, including tension in the cables for each drone, and the orientation of each drone.

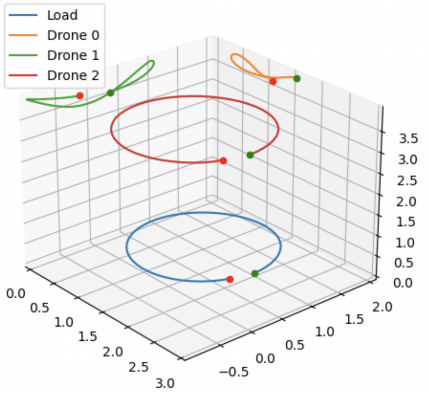

Fig. 13: Desired trajectory of each drone and the load when practical anchor points are used. The solver requires some fine tuning to smooth out the jagged trajectories.

The ability to add costs to the SNOPT solver sometimes made the resulting solutions more smooth, however this also made the solver take much longer. The process of adding costs was to start by finding a feasible solution without any costs, seeding the solver with this initial guess, then add the costs before re-solving. If the solution was too jagged, one of the costs could be manually turned on in the code, and the trajectory would be recalculated using that last trajectory to seed the solver.

IV. CONCLUSION AND FUTURE WORK

A. Simulations and Visualizations

Several errors were encountered while attempting to simulate the three drones with the tethers connected to the suspended load. Hence, future work would involve solving these errors and try to get a full visualization of the three drones and slung load following a trajectory generated by the differential flatness algorithm. The main problem after finding a stabilizable state about which to linearize for the LQR controller was that the "spatial forces applied to the system was not a number". When linearizing the system and inspecting the A and B matrices of the plant, it was hypothesized that the entries in these matrices may be causing a divide-by-zero error when solving for S in the algebraic Riccati equation. However, more work is needed to determine the actual root cause of the error.

Additionally, there were issues with adding an infinite horizon LQR on the deferentially flat trajectories since they did not end at a stationary point. Appending a stable stationary point at the end of the trajectory should eliminate these errors.

From here, the next step would be to represent the load and tethers more accurately. For the load, this means altering the SDF file for the load to better match its actual geometric, collision, and inertial properties. Additionally, the attachment points of the tethers would also be in more practical locations on the actual load rather than along the center plane of the load. For the tethers, this means changing the number of links to a number greater than one, probably more like ten or twenty. Doing this would make the dimension of the system state significantly larger, and finding a linearized state that is stabilizable would become significantly more difficult.

B. Differential Flatness

The orientation of the drones from the solver had a notable amount of jitter. This was a result of the use of a numerical solver instead of the closed-form solution. One way to fix this would be to include higher order derivatives of the dynamics equations as constraints in the SNOPT solver to make full use of the differential flatness properties of the system.

Other future changes include testing out different $Lambda$'s and seeing how constraining them impacts the drone and loads' trajectories.

REFERENCES

[1] Kenneth C. W Goh et al. "Aerial filming with synchronized drones using reinforcement learning". en. In: *Multimedia Tools and Applications* 80.12 (May 2021), pp. 18125–18150. ISSN: 1380-7501, 1573-7721. DOI: 10.1007/s11042-020-10388-5.

[2] Himanshu Pathak. "Use of Drones in Agriculture: Potentials, Problems and Policy Needs". en. In: *ICAR-NIASM* 300 (Aug. 2020), pp. 13+i.

[3] KarthikBalajee Laksham. “Unmanned aerial vehicle (drones) in public health: A SWOT analysis”. en. In: *Journal of Family Medicine and Primary Care* 8.2 (2019), p. 342. ISSN: 2249-4863. DOI: 10.4103/jfmpc.jfmpc_413_18.

[4] Jonathan Fink et al. “Planning and control for cooperative manipulation and transportation with aerial robots”. en. In: *The International Journal of Robotics Research* 30.3 (Mar. 2011), pp. 324–334. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364910382803.

[5] Nathan Michael, Jonathan Fink, and Vijay Kumar. “Cooperative manipulation and transportation with aerial robots”. en. In: *Autonomous Robots* 30.1 (Jan. 2011), pp. 73–86. ISSN: 0929-5593, 1573-7527. DOI: 10.1007/s10514-010-9205-0.

[6] Koushil Sreenath, Nathan Michael, and Vijay Kumar. “Trajectory generation and control of a quadrotor with a cable-suspended load - A differentially-flat hybrid system”. en. In: *2013 IEEE International Conference on Robotics and Automation*. Karlsruhe, Germany: IEEE, May 2013, pp. 4888–4895. ISBN: 978-1-4673-5643-5 978-1-4673-5641-1. DOI: 10.1109/ICRA.2013.6631275.

[7] Koushil Sreenath and Vijay Kumar. “Dynamics, Control and Planning for Cooperative Manipulation of Payloads Suspended by Cables from Multiple Quadrotor Robots”. en. In: *Robotics: Science and Systems IX*. Robotics: Science and Systems Foundation, June 2013. ISBN: 978-981-07-3937-9. DOI: 10.15607/RSS.2013.IX.011.

[8] Guofan Wu and Koushil Sreenath. “Geometric control of multiple quadrotors transporting a rigid-body load”. en. In: *53rd IEEE Conference on Decision and Control*. Los Angeles, CA, USA: IEEE, Dec. 2014, pp. 6141–6148. ISBN: 978-1-4673-6090-6 978-1-4799-7746-8 978-1-4799-7745-1. DOI: 10.1109/CDC.2014.7040351.

[9] Nasim Ullah et al. “A computationally efficient adaptive robust control scheme for a quad-rotor transporting cable-suspended payloads”. en. In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 236.2 (Feb. 2022), pp. 379–395. ISSN: 0954-4100, 2041-3025. DOI: 10.1177/09544100211013617.

[10] Uluhan Cem Kaya and Kamesh Subbarao. “Momentum Preserving Simulation of Cooperative Multirotors With Flexible-Cable Suspended Payload”. en. In: *Journal of Dynamic Systems, Measurement, and Control* 144.4 (Apr. 2022), p. 041007. ISSN: 0022-0434, 1528-9028. DOI: 10.1115/1.4053343.

[11] Keyvan Mohammadi, Shahin Sirouspour, and Ali Grivani. “Control of Multiple Quad-Copters With a Cable-Suspended Payload Subject to Disturbances”. en. In: *IEEE/ASME Transactions on Mechatronics* 25.4 (Aug. 2020), pp. 1709–1718. ISSN: 1083-4435, 1941-014X. DOI: 10.1109/TMECH.2020.2995138.

[12] Keyvan Mohammadi, Shahin Sirouspour, and Ali Grivani. “Passivity-Based Control of Multiple Quadrotors Carrying a Cable-Suspended Payload”. en. In: *IEEE/ASME Transactions on Mechatronics* 27.4 (Aug. 2022), pp. 2390–2400. ISSN: 1083-4435, 1941-014X. DOI: 10.1109/TMECH.2021.3102522.

[13] Hanafy M. Omar et al. “Recent advances and challenges in controlling quadrotors with suspended loads”. en. In: *Alexandria Engineering Journal* (Aug. 2022), S1110016822005245. ISSN: 11100168. DOI: 10.1016/j.aej.2022.08.001.

[14] Russ Tedrake. “Underactuated Robotics”. en. In: *Course Notes for MIT 6.832* (2023).

Appendix A
Code

The code used for this work can be found on Harvey Merton’s Github by following this link: https://github.com/hmer101/6.8210_project

Appendix B
Differential Flatness Results

This appendix includes output plots generated by the differential flatness solver for the trajectories of three drones. The trajectory may be viewed in meshcat by running the code in Appendix A. Here is the link to a video of the differential flatness running: https://youtu.be/qHHcxHXNx0I. The python file being run is: https://github.com/hmer101/6.8210_project/blob/main/diff_flatness/circle_tests/time_varying_lqr_circle_3drones_noload.py

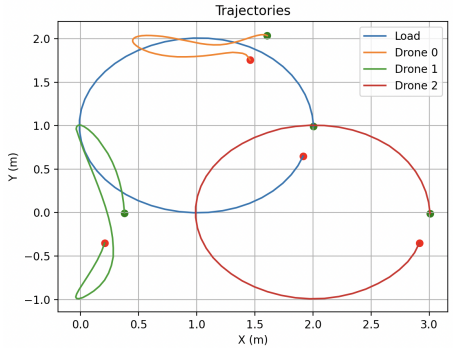

Fig. 14: Projection of the differential flatness output 3D trajectories in X and Y.

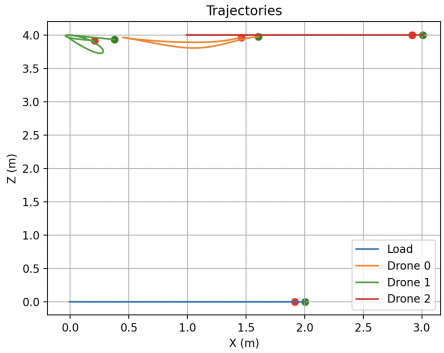

Fig. 15: Projection of the differential flatness output 3D trajectories in X and Z.

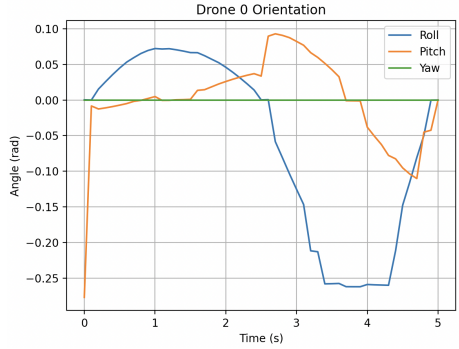

Fig. 16: Orientation of quadcopter 0 throughout the trajectory.

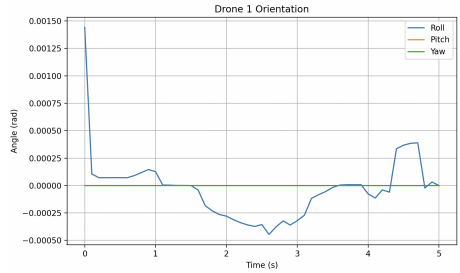

Fig. 17: Orientation of quadcopter 1 throughout the trajectory.

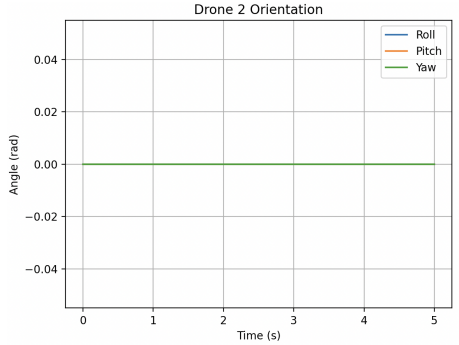

Fig. 18: Orientation of quadcopter 2 throughout the trajectory.

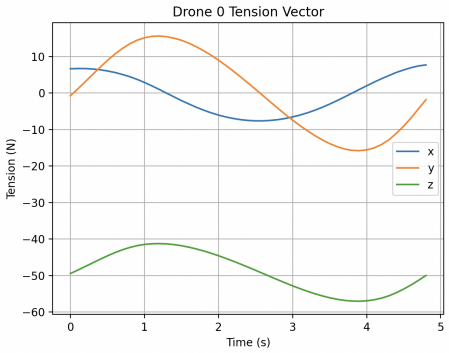

Fig. 19: Components of tension in the cable connecting quadcopter 0 to its anchor point on the load. Components are acting on drone pointing towards the load in the fixed coordinate system.

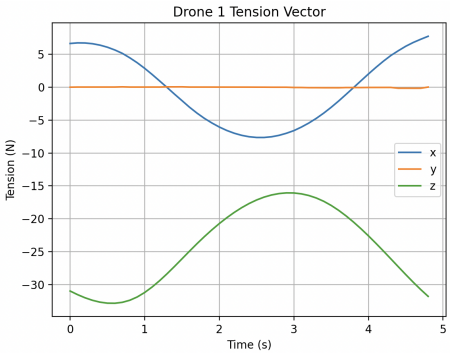

Fig. 20: Components of tension in the cable connecting quadcopter 1 to its anchor point on the load. Components are acting on drone pointing towards the load in the fixed coordinate system.

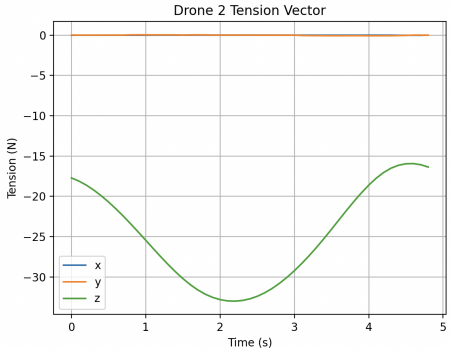

Fig. 21: Components of tension in the cable connecting quadcopter 2 to its anchor point on the load. Components are acting on drone pointing towards the load in the fixed coordinate system.