18.337 Final Project - Universal Differential Equations for Cable Tension Modelling in a Multi-Quadrotor Slung Load System

Harvey Merton

CONTENTS

I. INTRODUCTION

Quadrotors 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 quadrotors tend to be small and as such, can only carry payloads of limited sizes. Carrying cable-suspended payloads with multiple drones (see an example in Fig. 1) offers a way to overcome this barrier.

A. Current approaches to modelling quadrotor slung load dynamics

Model-based approaches can very successfully model traditional quadrotor dynamics. Such approaches generally take one of two methods to deal with additional slung loads: treating the additional mass as uncertainty and applying robust controller synthesis techniques, or adapting the underlying mathematical model [4]. The former approach is explored through adaptive robust control in [5], and through passivity-based control in [6, 7]. Although these methods show stability under a range of conditions, the lack of a complete explicit model for the load makes it harder to optimize for a minimum swing trajectory.

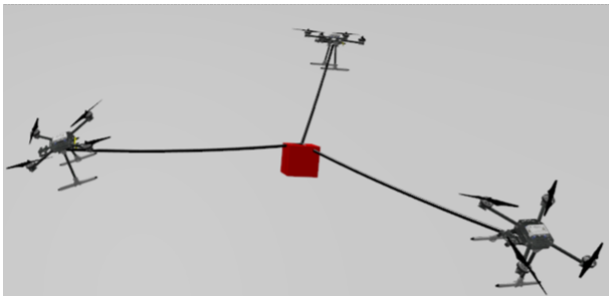


Fig. 1: Three x500 quadrotors and a slung load in a Gazebo simulation.

Modelling through adapting the underlying mathematical model is addressed by [8] and [9]. The latest of these, [9], 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 (which don't consider payload dynamics) that proceeded in [10, 11], thus modelling payload dynamics is useful.

A major problem with adapting the underlying mathematical model is the complexity required to capture general slung load dynamics. These include the case of zero cable tension (which [9] captures with hybrid system switching dynamics), non-rigid-body masses, cables with non-negligible mass, elastic cables, loads and cables where aerodynamic effects are non-negligible and cables not attached at a drone's center of mass. Ignoring cable dynamics (by assuming a massless cable) is particularly common due to the large variety and complexity of cable modelling techniques [12]. This simplification quickly becomes limiting when working with larger and heavier loads that require longer and stronger cables.

To address some of the difficulties in mathematical modelling, reinforcement learning has been applied in recent years [13, 14, 15, 16]. Although the results show some promise ([14] still notes significant instability under particular conditions), reinforcement learning poses its own suite of challenges, such as gathering a sufficient quantity of good quality training data, designing reward functions and tuning training parameters. Further, such approaches completely disregard the earlier modelling work proven to be successful for a range of cases.

B. Universal differential equations

Neural ordinary differential equations (NODEs), like Equation 1, are simply ordinary differential equations (ODEs) where the right-hand side is defined by a neural network.

$$\dot{u} = NN_\theta(u, t) \tag{1}$$

Universal differential equations (UDEs) are extensions of NODEs where differential equations are defined in full or part by a universal approximator (e.g. neural network) [17]. The Lotka-Volterra equations (Equation 2) provide a simple example, where U_1 and U_2 represent universal approximators.

$$\begin{aligned} \dot{x} &= \alpha x + U_1(\theta, x, y) \\ \dot{y} &= -\theta_1 y + U_2(\theta, x, y) \end{aligned} \tag{2}$$

This allows known physics models to be easily expanded to account for some effects that are difficult to model, whilst requiring a much smaller quantity of data to train than a traditional neural network (or alternative function approximator).

UDEs have been applied across a range of different fields to huge success [17], but have not yet become a cornerstone tool in mechanical engineering. This is will likely change in the coming years as researchers attempt to bridge the gap between model-based and model-free (such as reinforcement learning) dynamics and control techniques.

As discussed above, the multi-quadrotor slung load system is a perfect example of a mechanical system where researchers have jumped directly to model-free techniques to overcome limiting model-based assumptions. One of the main motivations of this project is to provide an example of a complex mechanical system where UDEs can overcome such limiting assumptions.

C. Goals and paper structure

The goal of this project is to capture the effects of cable tension on the drones and load in a multi-drone slung load system, using UDEs. The intention is to allow the prediction of tension vectors acting on both ends of the cables for cables with different masses. This combats the commonly-used 'massless cable' assumption (discussed in Section II-A) holding back model-based methods in slung load systems.

This paper first discusses the most popular mathematical model of the multi-drone slung load system. It points out the effect of the massless cable assumption, and how the formulation can be augmented with a UDE to capture the effects of cable mass. The discussion then moves to how the system is implemented in the Julia programming language, and how the embedded function approximators are trained. The training and prediciton results are presented for the full UDE system, as well as for the isolated function approximator. Discussion concludes with next steps for better training convergence. Key performance considerations and areas for future improvement are mentioned throughout.

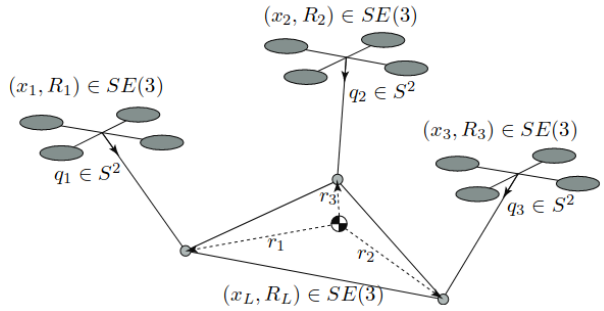


Fig. 2: Three quadrotors carrying a rigid body slung load [9].

II. Mathematical Modelling

A. Dynamics modelling

A complete dynamics model for the multi-drone rigid-body slung load system is developed in [9]. This model, based on the Euler-Newton method, is presented here. Fig. 2 illustrates the specific case when three drones are used, which is the case developed throughout the rest of this paper. The system of equations 3 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 [9].

$$m_i \ddot{x}_i = f R_i e_3 - m_i g e_3 + R_L T_i q_i \tag{3a}$$
$$J_i \dot{\Omega}_i + \Omega_i \times J_i \Omega_i = M_i \tag{3b}$$
$$m_L \ddot{x}_L = -\sum R_L T_i q_i - m_L g e_3 \tag{3c}$$
$$J_L \dot{\Omega}_L + \Omega_L \times J_L \Omega_L = \sum r_i \times -T_i q_i \tag{3d}$$

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

Several assumptions (also discussed in [9]) are made in deriving this system. These include:

1) The cables are massless.
2) The cables are constant length.
3) The cables are attached to the quadrotors at their centres of mass.
4) Aerodynamic drag acting on the quadrotors, load, and cables, is negligible.
5) The cables are always taut. The hybrid dynamics case that can handle slack cables is discussed in [9], but is not presented here.

The key assumption addressed in this paper is assumption 1 - the 'massless cable' assumption. A massless cable, when under positive tension, will draw a straight line between its two connection points. This means that the tension vectors acting on the connection points can be assumed to be equal and opposite. The effect of this assumption is seen in equations 3, where the tension vector $T_i q_i$ has equal magnitude but acts in the opposite direction on drone i (equation 3a) and the load (equations 3c and 3d).

In reality, a cable will have mass and thus be subject to the effects of gravity. This results in the cable hanging in an approximately caternary shape. Fig. 3 illustrates that the tension vectors acting on the drone and load connection points are no longer equal and opposite. This can have a large effect on the system dynamics and render the model useless for cables with greater mass.

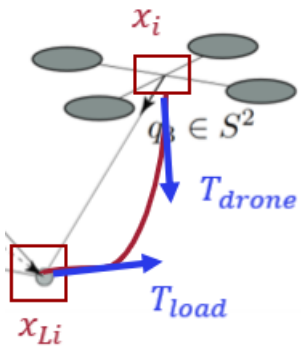


Fig. 3: Tension vectors at either end of a cable hanging in a caternary shape. The cable in question is connected to drone i at x_i and to the load at connection point x_{L_i}.

B. Cable tension modelling

We can use a neural network to capture the cable tension vectors acting on the drone and load respectively. Starting with the simple massless cable case, it is hypothesized that the model shown in Equations 4 and 5 can capture the tension vectors.

$$T_{drone} = NN_p(x_{i/L_i}, \dot{x}_{i/L_i}, \ddot{x}_{i/L_i}) \tag{4}$$

$$T_{load} = -T_{drone} \tag{5}$$

Equation 4 suggests that a neural network with parameters 'p' (refered to throughout as the 'tension-predicting neural network' and 'the neural network') can capture the tension vector (a 3x1 output) acting on the drone. It takes only the relative motion of the two cable attachment points as inputs to make the prediction: x_{i/L_i} is the position of the attachment point of a cable on drone i relative to its corresponding attachment point on the load L_i, similarly with relative velocity $\dot{x}_{i/L_i}$ and acceleration $\ddot{x}_{i/L_i}$. This gives a total of 9 inputs (each input is a 3x1 vector).

The relative motion of the cable attachment points are used to make the tension predictions independent of where the slung load system is relative to the inertial frame. This has a similar effect to data normalization. As the massless cable assumption is used, the tension vector acting on the load is equal and opposite to that acting on the drone (Equation 5).

To extend beyond the simple massless cable case, it is hypothesized that a separate neural network will be required to model the tension on the drone and the load respectively. To allow generalization between cables of different mass, the cable mass would also likely have to be an input. To capture the effects of varying other cable parameters such as elasticity, damping and aerodynamic effects (umong others), the key quantities defining these effects (perhaps cable stiffness, damping ratio, radius and coefficient of drag), would also have to be inputs. As a proof of concept however, this paper only deals with trying to replicate the simple massless case and so we proceed with the formulation in Equations 4 and 5.

It may be noted that it is possible to train the neural network proposed in Equation 4 without embedding it in a UDE. This is indeed done in Section IV-B. The problem with this however is that it is difficult to measure the exact orientation of the tension vectors T_{drone} and T_{load} on a physical system. Embedding the tension-approximating neural network in a UDE means we can train directly on drone and load trajectory data, which is much easier to obtain in a physical system or simulator.

III. Julia Implementation

All code for this project was implemented in the Julia programming language. Julia was used due to its support for fast scientific machine learning tools that are extremely useful for training and solving UDE systems. Some key aspects of the implementation are discussed below, and the full source code can be found on GitHub here: https://github.com/hmer101/18.337_project. All experiments involving timing were conducted on a Dell XPS 15 containing an Intel(R) Core(TM) i9-9980HK CPU clocked at 2.40GHz.

A. UDE system

The core of the project is the UDE system that implements the system of equations 3. These equations are implemented as a system of first order UDEs with the state space 'u' defined by Equation 7. $\vec{x_i}$ used here represents the state space of drone i, where $i \in 1, 2...n$ and $n = 3$, where all other terms are again defined in Table I.

$$\vec{x_i} = \begin{pmatrix} x_i & \dot{x}_i & \theta_i & \dot{\theta}_i \end{pmatrix}^T \tag{6}$$

$$\vec{u} = \begin{pmatrix} \vec{x_i} & ... & \vec{x_n} & x_L & \dot{x}_L & \theta_L & \dot{\theta}_L \end{pmatrix}^T \tag{7}$$

Julia's DifferentialEquations.jl package requires a function of the form $f1!(du, u, p, t)$ to pass into a differential equation solver. The mutating in-place form '$f1!$' is deliberately used over the out-of-place form so that du need not be re-allocated on every iteration of the solver. As many solver iterations are required to solve the system over a given time-span, this significantly reduces the number of allocations required during solving. This in turn significantly reduces the solve time, and so makes training tractable (many full solves over the entire timespan are required during training).

$$\dot{\vec{x_i}} = \begin{pmatrix} \dot{x}_i & \ddot{x}_i & \dot{\theta}_i & \ddot{\theta}_i \end{pmatrix}^T \tag{8}$$

$$\vec{du} = \begin{pmatrix} \dot{\vec{x_i}} & ... & \dot{\vec{x_n}} & \dot{x}_L & \ddot{x}_L & \dot{\theta}_L & \ddot{\theta}_L \end{pmatrix}^T \tag{9}$$

Equations 8 and 9 show where the drone and load acceleration terms appear in the du calculations. From Equations 3a, 3c and 3d, we can see that tension vector estimates ($T_i q_i$) appear when calculating $\ddot{x}_i$, $\ddot{x}_L$ and $\dot{\Omega}_L$. These estimates are found using Equations 4 and 5 defined earlier and thus represent the embedded universal approximator part of the UDE system.

$m_L \in \mathbb{R}$	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.
$\theta_i \in \mathbb{R}^3$	Orientation of the i^{th} quadrotor as roll-pitch-yaw (RPY) angles in the inertial frame. This is R_i converted to RPY Euler angles.
$\Omega_i \in \mathbb{R}^3$	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.
$x_{i/L_i} \in \mathbb{R}^3$	Relative position of the i^{th} quadrotor to its attachment point on the load (x_{L_i}) in the body-fixed frame of the load. Equivalent to $-L_i q_i$.
$r_i \in \mathbb{R}^3$	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

Again, Julia's DifferentialEquations solver limits the UDE system function to the form $f1!(du, u, p, t)$. As we only wish to find the sensitivity of the system to changes in parameters of the tension-approximating neural network, only the flattened parameters of the neural network are passed into $f1!$'s 'p'. This saves computation time when finding the gradient of $f1!$, which is very relevant when training the neural network; there is no need to find the sensitivity of the system to additional parameters we are not optimizing. All additional parameters required in the UDE system computation are defined in a mutable structure 'DroneSwarmParams' which wraps $f1!$. This results in the UDE system being defined as a callable structure: `(params::DroneSwarmParams)(du,u,p,t)`. A callable structure is used rather than a closure as it allows the UDE system to more easily be called in a variety of testing code blocks.

The UDE system's state space is comprised of 16 3x1 vectors in the three-drone case (Equations 6 and 7). Julia's differential equations solver requires 'u' and 'du' to be in flattened form. Rather than flattening the 16x3 matrix, array partitions were used. This allows the individual components to be accessed without having to unflatten parts of the array, which was particularly useful when rotation matricies, rather than roll-pitch-yaw Euler angles, were used to define the orientations of the drones and load.

B. Solver and loss functions

The UDE is solved using DifferentialEquations.jl's Tsit5 solver. Using the adaptive solver, it was found that timesteps were taken on the order of $1e^{-4}$ sec, which is magnitudes smaller than the timescale this system is expected to operate on (no less than $1e^{-3}$ s). This resulted in infeasibly long solve times of around 6 hours for a single trajectory 2.5 s long. Switching to a fixed timestep solver with a step of 0.1 s gave similar trajectory results (down to 2 decimal places), but with a solve time of 5 sec, thus the fixed time-step solver was used. It is of interest in the future to compare Julia's different solvers to verify the correctness of the solutions and potentially achieve faster solve times in this system.

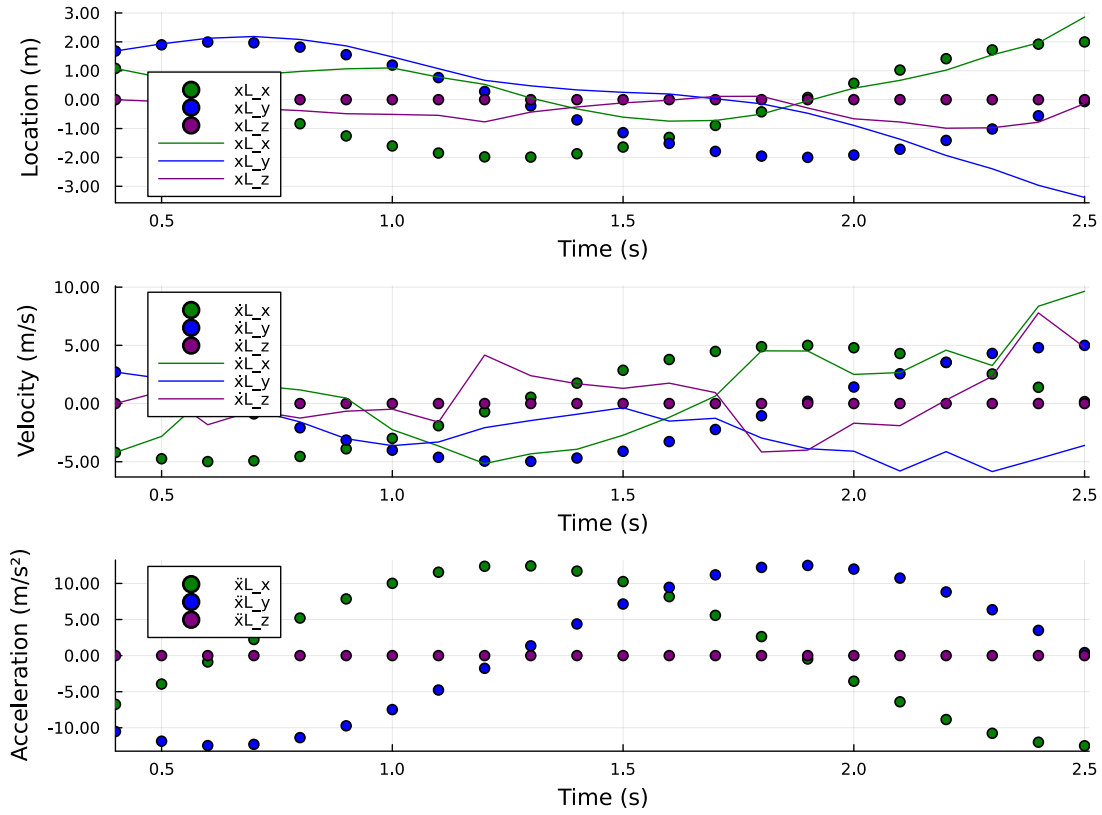
The loss function for the full UDE is defined as the mean squared error (MSE) between the full state space's trajectory data and the full state space predicted by the UDE solver. Similarly, MSE between the tension data and predicted tension over a trajectory is used when training the tension-predicting neural network directly.

C. Training data generation

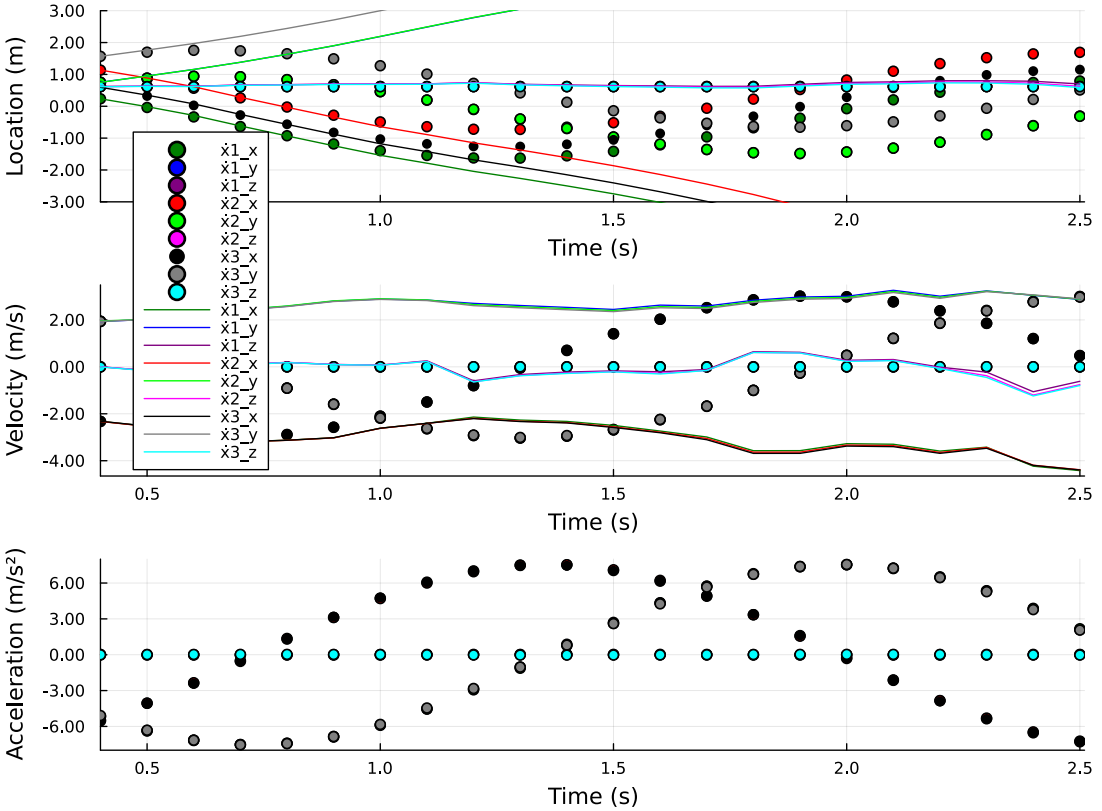
It was initially intended that trajectory training data would be generated in simulation. However, the simulator being built in parallel is not yet in a functional form. The data is instead generated using a prescribed load trajectory and, using the property of differential flatness discussed in [9], along with the system of dynamics equations 3, the corresponding drone trajectories can be solved. This gives a full time-history of the state space required to train the neural network embedded in the full UDE. Saving the cable tensions at the same datapoints as as the trajectories, a full set of training data becomes available for training the neural network directly, rather than when it is embedded in the UDE.

The load trajectory is selected to be the load moving in a circle of constant 2 m radius with a period of 2.5 s. The load does not rotate relative to the inertial frame during motion, thus giving a constant linear acceleration towards the center of the revolution, and no angular velocity or acceleration. This is the simplest case containing some varying cable tensions throughout the trajectory. The load and drones' linear motions can be found as datapoints in Figs. 4a and 4b respectively. Similarly, the relative motion of the cable attachment points and the corresponding cable tensions can be found in Figs. 5a and 5b. All angular motions (orientation, angular velocity and angular acceleration) of the load and drones remain at zero for the full extent of the trajectory. This information fully describes the slung load and drone system trajectory.

As only the drone positions are directly constrained by the cable, data for the drones' linear velocities and accelerations

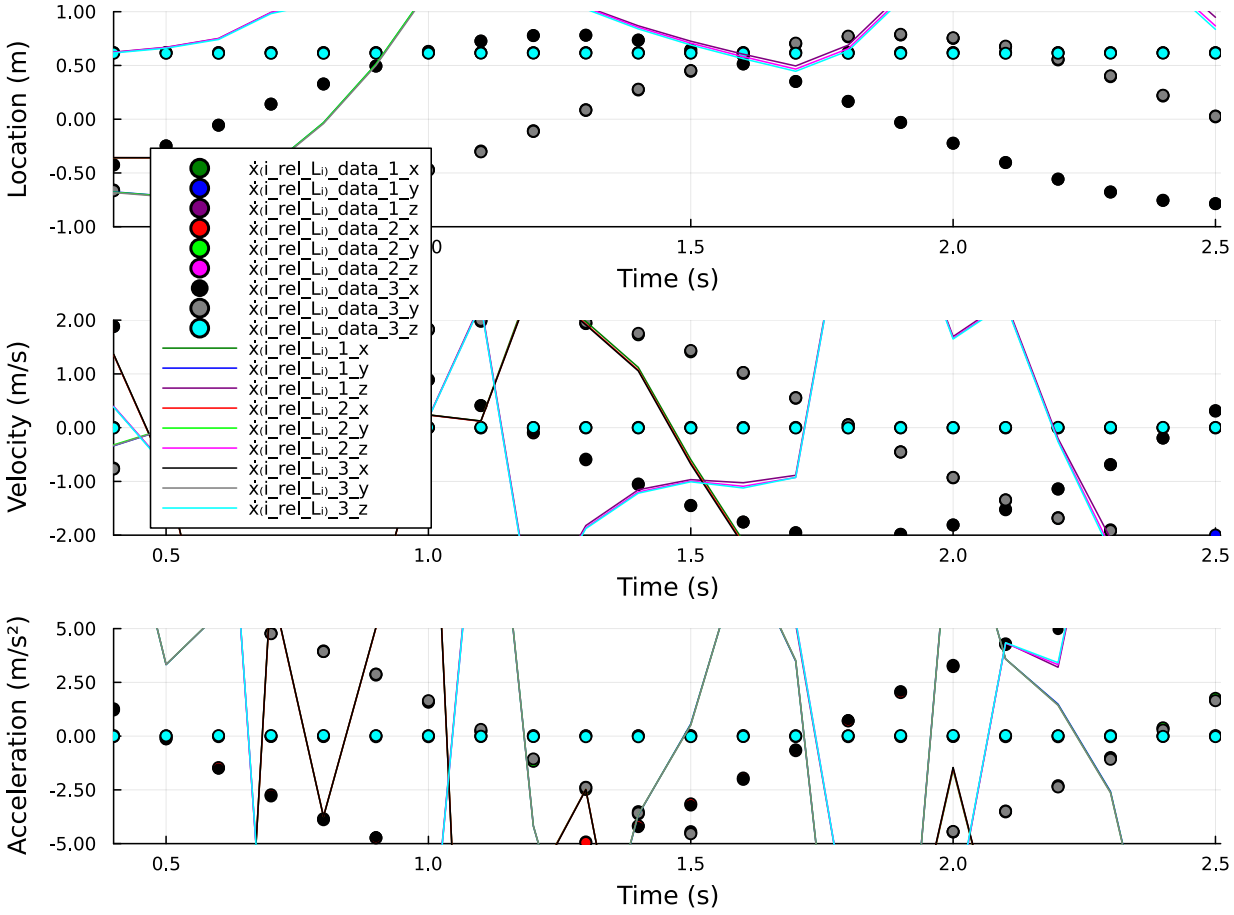


(a) Linear trajectory data and predictions - load.

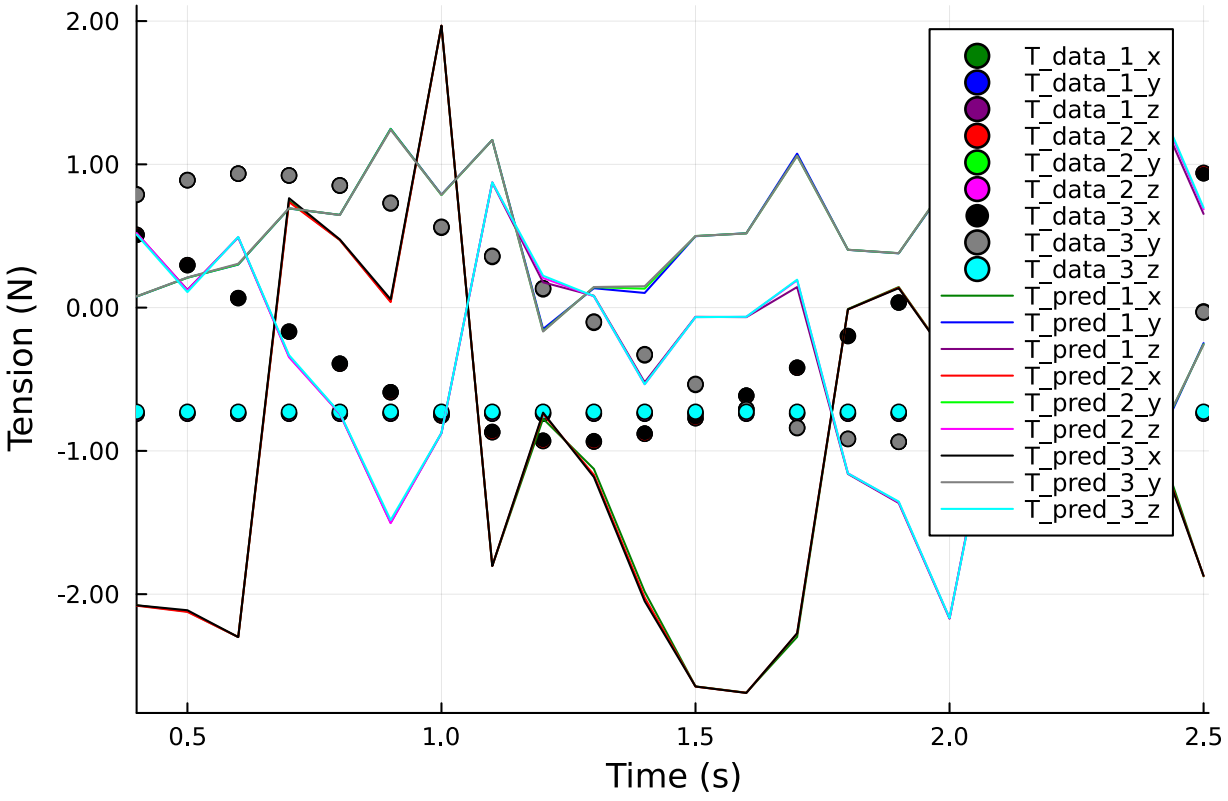


(b) Linear trajectory data and predictions - drones. Legend corresponds to all graphs - excluding the dot for the first graph, and adding a dot for the last graph.

Fig. 4: Training data (points) and post-training predictions (lines) for a three-drone rigid-body load system. The load travels anti-clockwise in a circle without vertical motion. The drones remain arranged in a way such that neither the load nor the drones rotate over the trajectory. The first 'x' in the labels represents state while L, 1, 2 & 3 index the load and the drones. Data is split into x,y and z components relative to the inertial frame as identified by the trailing 'x','y' or 'z'. Data points for cables that appear to be missing are directly under the corresponding data points for another cable.



(a) Relative motion of cable attachment points.



(b) Cable tensions throughout trajectory.

Fig. 5: Cable tension and relative cable attachment point motion data and predictions from the same trajectory as Fig. 4. Predictions come from training the neural network embedded in the UDE.

has to be generated by a finite backwards difference (otherwise the dynamics could be over-constrained). This gives poor initial estimations of drone velocity for the first timestep, and poor estimates of the drone acceleration for the first two timesteps. Rather than hard-coding the initial values for these quantities, the training data is simply trimmed to exclude the first three timesteps.

IV. Results and Discussion

Initially, for the reasons discussed in Section II-B, the neural network was trained whilst embedded in the UDE system. When this did not converge after extensive debugging, the cable tension-predicting neural network was trained directly on tension, and relative motion of the cable attachment points, data. The results of both approaches are discussed below.

A. UDE system

Julia's Optimization.jl package is used to train the system due to the relative ease of switching between methods of finding gradients in the back end. This is useful because it was initially intended to use automatic differentiation (AD) to find the gradients required for training. However, it was discovered too late that ForwardDiff, ReverseDiff, Zygote and Tracker all have difficulty functioning with pre-allocated caches used in a function solved by a UDE solver. As pre-allocated caches are included in the callable structure that surrounds the UDE system, AD was unable to be used without the use of the PreallocationTools.jl package. Implementing this when the issue was discovered would have taken too long, so it is left for future work.

Instead, FiniteDiff.jl was used to find the gradient of the loss function (with respect to neural network parameter changes), using finite differences. This is significantly slower and less accurate than the ideal source-to-source AD from Zygote.jl.

The tension-predicting neural network is modelled with 1 hidden layer containing 20 nodes. When this was trained using the ADAM optimizer, a learning rate of 0.01 and a maximum of 1000 iterations, convergence was not reached (see Fig. 6). Observing the plateau at the end of the loss history, it was hypothesised that perhaps the training had stabilized and needed more iterations to converge. Increasing the iteration cap to 10,000 however, did not help. Increasing the learning rate to 0.02 lead to an even further unstable loss history, and decreasing it lead to a similar shallow training gradient as seen at the end of Fig. 6. Increasing the number of hidden layers and hidden nodes did not help either, and given the results in IV-B, it is not expected to do so.

Testing the gradient found from the FiniteDiff.jl and that found from the FiniteDifference.jl packages, it is found that a significant discrepancy exists (on the order of 1-10% for different parameters). This is concerning and suggests that the finite difference methods may have significant error build-up that results in lack of accurate gradients. This would be the leading theory for lack of convergence if not for the discussion at the end of Section IV-B. It will however be addressed when implementing the preallocationTools fix discussed above as AD avoids finite differencing's error accumulation.

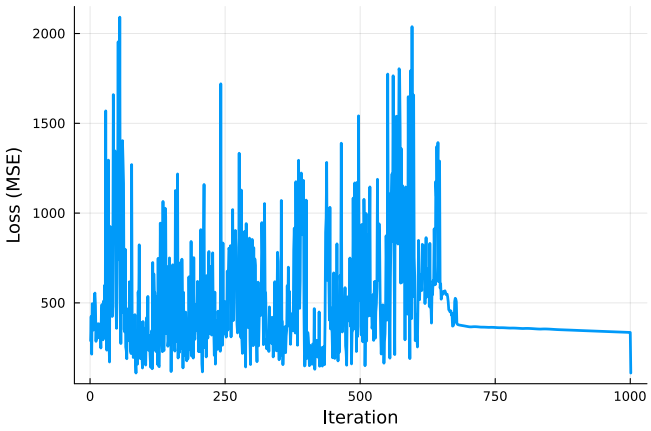


Fig. 6: Loss history from training the neural network embedded in the UDE.

Training takes on the order of 10 minutes when trained on the CPU with GPU training not attempted due to lack of initial convergence. Comprehensive benchmarking will be completed once convergence is achieved, and reverse-mode AD is able to be performed on the UDE (training is expected to be significantly faster).

The trajectory predictions using the non-converged neural network tension estimator are shown in Fig. 4.

B. Directly training tension approximation function

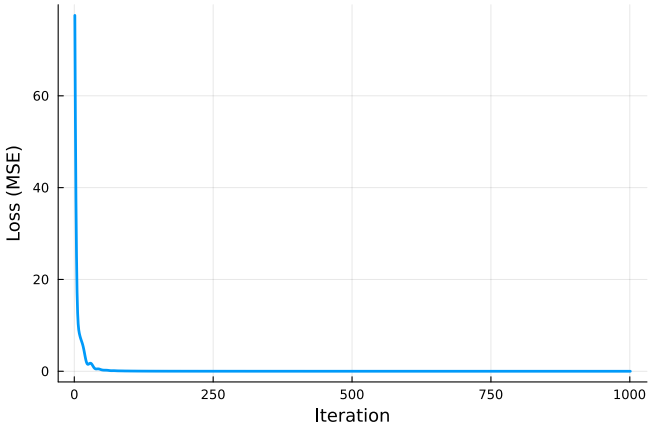
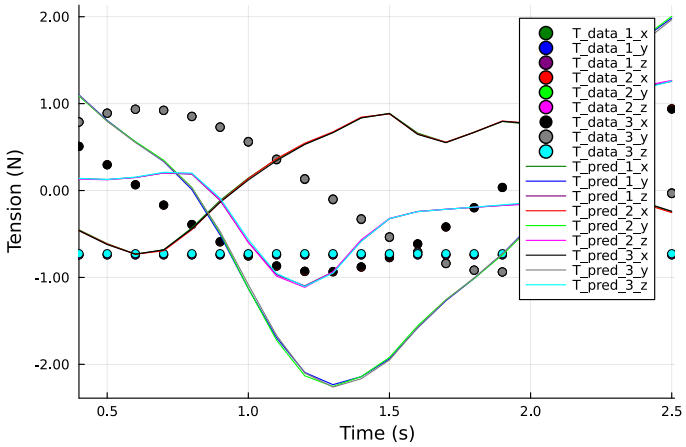


Fig. 7: Loss history while training the neural network directly.

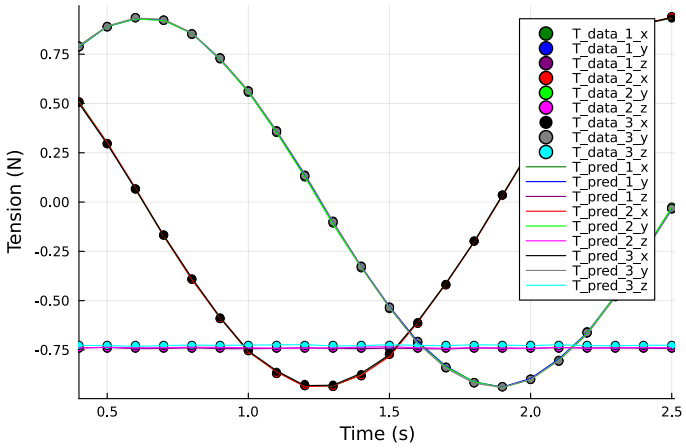
It was hypothesised that perhaps the tension-approximating neural network was unable to capture the tension data with only knowledge of the relative motion of the cable attachment points. To test this hypothesis, the tension-predicting neural network was trained directly on the cached tension and relative cable attachment point motion data. The neural network remained the same as used in Section IV-A (1 hidden layer with 20 nodes), as did the hyperparameters. Without the callable structure's caching causing issues, source-to-source reverse-mode AD was able to be used to find the gradients for training. This, combined with the lack of need to solve the UDE system,

resulted in a training time of under 5 s. Convergence is shown to occur in < 200 iterations in Fig. 7.

Fig. 8 shows the tension prediction results before and after training, showing that the neural network is successfully able to predict cable tensions given knowledge of only the relative motion of the cable connection points. To cement this proof, the trained neural network would need to be tested on tension data generated from another trajectory. This has not yet been performed as the main purpose of training the neural network directly was to attempt to debug the UDE system. This will be completed in the future however.



(a) Cable tensions throughout trajectory - pre-training.



(b) Cable tensions throughout trajectory - post-training.

Fig. 8: Cable tension and relative cable attachment point motion data and predictions from the same trajectory as Fig. 4. Trained predictions come from training the neural network directly.

Even when using this trained neural network, the full UDE is unable to correctly predict drone and load trajectories. This suggests that the error is not in the neural network training, but rather in the UDE system itself, or in the solver. Perhaps the solver is incorrectly selected for the task, or the system is chaotic. Further investigation into both of these is required.

It is interesting to note that when warm-starting the UDE system training with the already trained neural network parameters, convergence was again not achieved. This is expected as it has been determined that an issue exists in the UDE solver which affects the training process (the UDE is solved when calculating the loss, which is required to train).

V. Conclusions

Universal Differential Equations provide a new avenue to remove limiting assumptions in model-based dynamics and controls applications. This paper presents a method to address the massless cable assumption in a multi-quadrotor slung load system. It was shown that a simple (1 hidden layer with 20 nodes) neural network is capable of predicting cable tensions in the massless cable case, given only the relative position, velocity and acceleration of the attachment points of the cable. When this neural network was embedded in a full UDE system to predict load and drone trajectories however, the estimates were wild. This suggests that this methodology shows some promise, and with more work on selecting UDE solvers and debugging the UDE implementation, may be able to provide good trajectory estimations. This work is intended to be continued and should it be completed, will provide a practical full-scale example of how UDEs can be successfully used in mechanical systems.

VI. Future Work

Future work includes:

- Debugging the UDE system to achieve accurate trajectory predictions with the trained neural network. This will involve investigating if the system is chaotic, if the correct differential equation solver is being used (including adaptive solver and step size parameters), and if the rest of the implementation is performing as expected.
- Use Julia's preallocationTools to allow automatic differentiation to be used to find gradients of the UDE system. This will give faster and more accurate gradients for training.
- Investigate if the trained neural network can generalize predictions to unseen trajectories.
- Training across varied cable masses to capture the non-massless cable case.
- Streamlining implementation to make extensions and maintenance easier.

References

[1] Kenneth C. W Goh et al. "Aerial filming with synchronized drones using reinforcement learning". 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". In: *ICAR-NIASM* 300 (Aug. 2020), pp. 13+i.

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

[4] Aldo Enrique Vargas Moreno. “Machine Learning Techniques to Estimate the Dynamics of a Slung Load Multirotor UAV System”. In: (2017).

[5] Nasim Ullah et al. “A computationally efficient adaptive robust control scheme for a quad-rotor transporting cable-suspended payloads”. 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.

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

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

[8] Koushil Sreenath, Nathan Michael, and Vijay Kumar. “Trajectory generation and control of a quadrotor with a cable-suspended load - A differentially-flat hybrid system”. In: *2013 IEEE International Conference on Robotics and Automation*. 2013 IEEE International Conference on Robotics and Automation (ICRA). 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.

[9] Koushil Sreenath and Vijay Kumar. “Dynamics, Control and Planning for Cooperative Manipulation of Payloads Suspended by Cables from Multiple Quadrotor Robots”. In: *Robotics: Science and Systems IX*. Robotics: Science and Systems 2013. Robotics: Science and Systems Foundation, June 23, 2013. ISBN: 978-981-07-3937-9. DOI: 10.15607/RSS.2013.IX.011.

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

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

[12] Naijing Lv et al. “A review of techniques for modeling flexible cables”. In: *Computer-Aided Design* 122 (May 2020), p. 102826. ISSN: 00104485. DOI: 10.1016/j.cad.2020.102826.

[13] Aleksandra Faust et al. “Automated aerial suspended cargo delivery through reinforcement learning”. In: *Artificial Intelligence* 247 (June 2017), pp. 381–398. ISSN: 00043702. DOI: 10.1016/j.artint.2014.11.009.

[14] Xiaoxuan Li, Jianlei Zhang, and Jianda Han. “Trajectory planning of load transportation with multi-quadrotors based on reinforcement learning algorithm”. In: *Aerospace Science and Technology* 116 (Sept. 2021), p. 106887. ISSN: 12709638. DOI: 10.1016/j.ast.2021.106887.

[15] Shuai Li and Damiano Zanotto. “Multi-UAV Cooperative Transportation Using Dynamic Control Allocation and a Reinforcement Learning Compensator”. In: *Volume 9: 17th International Conference on Multibody Systems, Nonlinear Dynamics, and Control (MSNDC)*. ASME 2021 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Virtual, Online: American Society of Mechanical Engineers, Aug. 17, 2021, V009T09A034. ISBN: 978-0-7918-8546-8. DOI: 10.1115/DETC2021-71797.

[16] Fotis Panetsos, George C. Karras, and Kostas J. Kyriakopoulos. “A Deep Reinforcement Learning Motion Control Strategy of a Multi-rotor UAV for Payload Transportation with Minimum Swing”. In: *2022 30th Mediterranean Conference on Control and Automation (MED)*. 2022 30th Mediterranean Conference on Control and Automation (MED). Vouliagmeni, Greece: IEEE, June 28, 2022, pp. 368–374. ISBN: 978-1-66540-673-4. DOI: 10.1109/MED54222.2022.9837220.

[17] Christopher Rackauckas et al. *Universal Differential Equations for Scientific Machine Learning*. Nov. 2, 2021. arXiv: 2001.04385[cs,math,q-bio,stat].