Nonlinear Model Predictive Control of an Aerial Manipulator using a Recurrent Neural Network Model

Gowtham Garimella
Department of Mechanical Engineering
Johns Hopkins University
3400 N Charles Str, Baltimore, MD 21218, USA
ggarime1@jhu.edu

Mathew Sheckells
Department of Computer Science
Johns Hopkins University
3400 N Charles Str, Baltimore, MD 21218, USA
mshecke1@jhu.edu

Marin Kobilarov
Department of Mechanical Engineering
Johns Hopkins University
3400 N Charles Str, Baltimore, MD 21218, USA
marin@jhu.edu

Abstract: The goal of this work is to control an aerial manipulator system which consists of an Unmanned Aerial Vehicle (UAV) platform equipped with an articulated robotic arm, through model-predictive control based on a data-driven dynamical model. The learned model captures both the internal closed loop dynamics of the UAV autopilot as well as the servo motor control logic. At the core of the model lies a Recurrent Neural Network (RNN) architecture combined with a feedforward model to produce the linear and angular accelerations applied on the robot. These accelerations are then integrated in a standard manner to obtain the next robot state. The learned RNN architecture is then leveraged in a Nonlinear Model Predictive Control (NMPC) framework that accounts for the sparsity inherent to the trajectory optimization problem. The NMPC optimization has been successfully implemented onboard a DJI Matrice UAV platform with a custom made two-link arm to track desired reference trajectories.

Keywords: aerial manipulation, control, learning, RNN, MPC, UAV

1 Introduction

We consider the control design for accurate trajectory following of an Unmanned Aerial Vehicle (UAV) equipped with a multi degree-of-freedom (DOF) arm. Such a robotic system is known as an aerial manipulator, and could have potential application in package delivery [1], bridges and furnace inspection [2, 3, 4], cooperative transportation [5], stippling [6], pruning tree branches [7], and aerial sampling [8]. Such applications often require precise control of the end-effector position. For example, small errors in the end-effector position can result in failure to pick up a package or retrieve a sample.

The control of an aerial manipulator is complicated primarily due to the interactions between the multi-DOF arm and the UAV platform. Several control techniques have been developed in the past for controlling the aerial manipulator [9, 10, 11]. These controllers usually consider the inputs to the

2nd Conference on Robot Learning (CoRL 2018), Zrich, Switzerland.

system as the body torques and the body thrust applied by the UAV platform and the joint torques applied by the arm. In practice, off-the-shelf UAVs are controlled using an on-board autopilot such as DJI A3 [12] which controls the orientation of the UAV and a scaled thrust along the body z-axis of the UAV. Similarly, the arm is controlled using a servo motor that regulates the joint angles or joint velocities. The control logic of the autopilot and the servo motors is usually not known to the user for modeling purposes. Even if the logic is known, the exact forces and torques applied to the system are unknown since the controllers only use an approximate actuator model [13].

In this work, we employ a Recurrent Neural Network (RNN) to model the complete coupled aerial manipulator dynamics. Using an RNN allows for modeling the unknown control logic and the actuator dynamics based on sensor measurements. The RNN model is trained by piloting the aerial manipulator manually and collecting sensor data along the piloted trajectories. The RNN model is augmented with a feedforward model that accounts for the well-known rigid body kinematics and a simplified second order PD control model. The augmented RNN model is able to predict the aerial manipulator position accurately with an RMS error of 5 centimeters for an open-loop prediction horizon of 1 second based on a typical test dataset. The trained RNN model is then used in a Nonlinear Model Predictive Control (NMPC) framework to control the aerial manipulator. We experimentally compare the performance of the RNN model against a standard feedforward model in tracking a spiral reference trajectory for the UAV and sinusoidal reference trajectory for the arm.

1.1 Related Work

Aerial manipulator systems have been studied extensively in the past. Orsag et al. [14] showed design and control of a UAV attached with a multi-DOF manipulator. Kondak et al. [15] showed the control of an industrial 7DOF arm attached to a helicopter system. Cooperative transportation of aerial system has also been addressed in [16]. Kondak et al. [17] further showed cable transportation of load using three helicopters simultaneously. Nguyen et al. [18] simplified the cooperative transportation problem by decoupling the rotational dynamics of individual quadrotors using a ball joint attached at it's base. Most methods based on traditional non-adaptive control require careful tuning and in general lack robustness to large changes in the dynamics, for instance due to wind gusts, propeller down-wash, or contacts.

NMPC optimization is a control scheme that solves an optimization problem at every step by propagating the system dynamics and minimizing a user defined cost function along the trajectory. It accounts for system constraints and could handle changing system dynamics. NMPC optimization for a higher dimensional system such as an aerial manipulator is computationally expensive and has been initially limited to simulations. More recently, Nikou et al. [19] showed cooperative transportation of a load using multiple quadrotors while avoiding collisions and singularities using simulated dynamics. Neunert et al. [20] and Lunni et al. [21] showed NMPC optimization onboard a multirotor vehicle. The effectiveness of the NMPC methods depend on the accuracy of the model used. There is a non-trivial effort involved in identifying the model required for the NMPC optimization and updating the system parameters as the aerial manipulator interacts with the environment. This work considers the constructing such complex models using a RNN to predict the system motion.

The application of neural networks to aerial robots has generally been limited to vision-based recognition and path planning applications (e.g. Carrio et al. [22], Maciel-Pearson and Breckon [23], Heylen et al. [24]). Recently, end-to-end learning models have been used to control robot systems based on raw sensor data (Kelchtermans and Tuytelaars [25]). These methods are difficult to generalize to additional sensing modalities and to handling changes in the system dynamics. In this work, we use a RNN model to learn the dynamics of the aerial manipulator and use NMPC optimization for controlling the robot based on the predicted dynamics. This two-stage approach allows us to reliably verify the predictive capability of the RNN before using the controller thereby reducing the chance for potentially unsafe robot behavior.

Deep NMPC which combines a deep RNN with NMPC optimization has been proposed by Lenz et al. [26] for a robot cutting task. Unlike the RNN proposed in that work, we employ a much simpler model using IMU, joint angles, and thrust sensor data, and also combining the *unknown* learned dynamics with a *known* feedforward model for which we also learn the gains. Further, this model uses a known hidden state which can be estimated using traditional estimators. It also allows for substitution of sensors without the need to retrain the network.

Other related work includes the use of an RNN model in an NMPC scheme to control the steering dynamics of a passenger vehicle in [27]. The proposed RNN model predicts the error between predicted feedforward state and the measured state. In contrast, the RNN architecture used here produces accelerations as output which is integrated through a separate integration scheme. This makes the output trajectories smoother and thereby the controls produced are also smoother.

1.2 Contributions

We propose a novel RNN architecture that generates the accelerations in a robot state and integrates the accelerations to produce the next robot state. The integration of the accelerations is performed in a separate integration stage that takes as inputs the robot state, control and time-step of integration. We then use a semi-implicit integration scheme to maximize accuracy in prediction. The RNN architecture is augmented with a feedforward model to increase the predictive capacity of the model considerably with only a small increase in the number of parameters. A smaller-size model is also critical for fast real-time control optimization.

The resulting RNN is employed as a predictive model in an NMPC framework to track reference trajectories. A second-order stage-wise Newton [28] trajectory optimization method is employed which has a $O(N)$ complexity where N is the number of trajectory time-steps. The resulting algorithm can operate at 100 Hz using a 2 layer RNN model with 16 and 8 nodes running on an embedded i5 computer onboard the UAV platform. We then perform empirical evaluation of the control scheme on a UAV platform tracking spiral reference trajectories. The NMPC scheme achieved an approximate position accuracy of 0.1 meters and joint angle accuracy of 0.1 radians when tracking spiral reference trajectories for the UAV and sinusoidal trajectories for the joint arm simultaneously.

2 Modeling dynamics

We employ a quadrotor UAV platform equipped with 2-dof manipulator arm as shown in Fig. 1a. The quadrotor platform is an underactuated system consisting four co-axially aligned propellers. These propellers can control independently the three body torques around the principal axes of the body and thrust along the body z-axis. We assume that the quadrotor's orientation given by Euler angles ($\xi \in \mathbb{R}^3$) and the thrust $f_z \in \mathbb{R}$ are controlled using an autopilot module and the joint angles of the arm ($r \in \mathbb{R}^2$) are controlled using servo motors attached to the two joints.

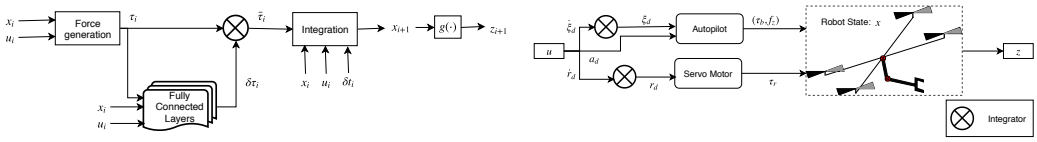
We assume the autopilot dynamics is a second order system i.e. the autopilot applies body torques ($\tau_b \in \mathbb{R}^3$) based on the commanded Euler angle rates ($\xi_d \in \mathbb{R}^3$) and the desired Euler angle rates ($\dot{\xi}_d \in \mathbb{R}^3$). It is also assumed that the thrust f_z applied by the autopilot is directly related to commanded thrust $a_d \in \mathbb{R}$. Hence, we assume the inputs to the autopilot system are $(\dot{\xi}_d, a_d)$ and treat ξ_d as part of the state. Similarly, assuming the servo model is a second order system applying joint torques $\tau_r \in \mathbb{R}^3$, the inputs are chosen to be desired joint angle rates $\dot{r}_d \in \mathbb{R}^2$ and the desired joint angles $r_d \in \mathbb{R}^3$ are assumed to be part of the state.

The quadrotor platform is attached with motion capture markers that are tracked using motion capture cameras which provide the position $p \in \mathbb{R}^3$ and the orientation ξ in an inertial frame. The servo motors provide feedback in terms of joint angles $r \in \mathbb{R}^2$. The schematic of the aerial manipulator shown in Fig. 1a shows the inputs and outputs associated with the aerial manipulation model. The combined sensor measurements $z \in \mathbb{R}^8$ and the controls $u \in \mathbb{R}^6$ are shown in (**??**).

$$z = [p, \xi, r]^T, \quad u = [a_d, \dot{\xi}_d, \dot{r}_d]^T \tag{1}$$

2.1 Network Architecture

The goal of the RNN is to predict the outputs at the next step z_{i+1} (i.e. the sensor measurements) given the control at the current step u_i and the hidden state x_i where i denotes the sequence index which in our context denotes the time elapsed. We want to use as much prior information about the model as possible to minimize the amount of training data required and reduce the size of networks. Smaller networks enable lower computational effort during NMPC optimization. To use prior information known about the model, the hidden state x is chosen manually to denote all possible feature that are expected to provide z_{i+1}. For the aerial manipulation system, the state x is selected to denote the full dynamical state of the system, i.e. the position of the quadrotor p, the velocity of the



(a) (b)

Figure 1: Schematic (a) shows the neural network architecture; Schematic (b) shows the aerial manipulation system with inputs u, outputs z, and robot state x.

platform $v \in \mathbb{R}^3$, the Euler angles ξ, the rate of Euler angles $\dot{\xi}$, the commanded Euler angles ξ_d, the joint angle r, the joint velocities $\dot{r}$, commanded joint angles r_d and a scaling coefficient on the commanded thrust denoted as $k_t \in \mathbb{R}$. The scaling coefficient is a part of the feedforward model that predicts the body z-axis acceleration given the commanded thrust as shown in (**??**). The overall state is given as

$$x = [p, \xi, v, \dot{\xi}, \xi_d, r, \dot{r}, r_d, k_t] \tag{2}$$

The mapping between the hidden state and the sensor measurements is denoted by $g(\cdot)$ and simply selects the correct sensor channels from the hidden state, i.e. $z = g(x) = [p, \xi, r]$. Since the hidden state is manually selected, its propagation is non-trivial and cannot be accomplished using a small number of fully connected layers. We can rely on prior information to propagate the hidden state and rely on fully connected layers to only learn the residual dynamics. The prior information is encoded into the architecture in two phases: "Force prediction" and "Integration" as shown in Fig. 1. In the first phase, the hidden state and the control are used to generate the resulting accelerations produced on the system. These accelerations are corrected using fully connected layers to account for unmodeled dynamics. The corrected accelerations are then integrated using a semi-implicit integration scheme. The integration phase of the model only depends on the time step of integration and usually does not have parameters that need to be trained.

Using the network to predict the perturbations in forces and torques makes the system trajectories smooth up to second order which is necessary for smoother control input to the system. Further, using integration which is decoupled from the acceleration prediction allows the same trained network to be used with different time steps during integration. In addition, the integration phase of the model can also incorporate limits on accelerations and velocities that are known before hand. For example, the joint velocities of the arm are bounded by 0.7 radians per second by the servo controller. This can be easily incorporated into the integration phase and does not require learning the same.

2.2 Force prediction model

The prior information about the quadrotor system and servo control is used to formulate a model that predicts inertia normalized forces, i.e. accelerations, that are required to propagate current state and control. For producing the angular accelerations on the quadrotor and joint accelerations on the arm, a second order PD control loop is applied between both the commanded and observed Euler angles ξ and the commanded and observed joint angles r. The acceleration on the quadrotor platform is obtained by scaling the commanded thrust and compensating for gravity ($g = [0, 0, 9.81]^T$). The overall feedforward dynamics can be written as

$$a = k_t a_d \bar{z} - g, \tag{3}$$

$$\ddot{\xi} = -k_{p_\xi}(\xi - \xi_d) - k_{d_\xi}(\dot{\xi} - \dot{\xi}_d), \tag{4}$$

$$\ddot{r} = -k_{p_r}(r - r_d) - k_{d_r}(\dot{r} - \dot{r}_d), \tag{5}$$

where $\bar{z}$ denotes the body z axis of the quadrotor in inertial frame which is obtained from the Euler angle ξ. The concatenated acceleration vector is denoted as $\tau = [a, \ddot{\xi}, \ddot{r}] \in \mathbb{R}^6$. The unknown parameters in the model are the proportional and derivative gains for Euler angles ($k_{p_\xi} \in \mathbb{R}^3, k_{d_\xi} \in \mathbb{R}^3$) and joint angles ($k_{p_r} \in \mathbb{R}^2, k_{d_r} \in \mathbb{R}^2$). These parameters are optimized along with the fully connected layer weights and biases. The force prediction model does not include the interactions between the arm and UAV since such a model depends on moment of inertia of the platform which is not observable without having access to the joint torques and body torques applied.

2.3 Residual dynamics

The accelerations produced by the feedforward model are not accurate since it incorporates only a simplified model of the actual dynamics and neglects the interactions between the quadrotor and the arm. The accuracy of the feedforward model can be improved by adding fully connected layers that predict the difference between the actual accelerations and the accelerations generated by the feedforward model, denoted by $\delta\tau \in \mathbb{R}^6$. The input to the network is given by the feedforward accelerations τ, the current state x, and controls u as shown in Fig. 1. It is necessary for us to scale the inputs before passing them into the fully connected layers to avoid saturating the neural network activation functions. We use batch normalization to automatically figure out the scale of the inputs. It is also safe to assume that the horizontal position of the quadrotor does not affect the acceleration of the quadrotor. Hence the horizontal position is neglected before passing the state into the fully connected layers. We use dropout layers on the intermediate fully connected layers and the residual correction $\delta\tau$ to ensure the feedforward dynamics is also trained even if the network produces noisy corrections.

2.4 Integration

The integration block integrates the corrected accelerations $\bar{\tau} = \tau + \delta\tau$ for a specified time step to obtain the state of the robot at next step. We used a semi-implicit integration scheme for integrating the accelerations. Under this scheme, the velocities at the next step are found by integrating the accelerations, and the averages of the current and predicted velocities are used to integrate the positions forward. A similar approach is used for integrating joint angles as shown below:

$$\dot{r}_{i+1} = \dot{r}_i + \delta t_i \ddot{r}_i, \tag{6}$$

$$\dot{r}_{i+1} = \min(\dot{r}_{i+1}, \dot{r}_{max}), \tag{7}$$

$$r_{i+1} = r_i + \delta t_i \frac{1}{2} \left(\dot{r}_{i+1} + \dot{r}_i \right), \tag{8}$$

where δt_i denotes the time step of integration and $\dot{r}_{max} \in \mathbb{R}^2$ denotes the maximum joint velocity. The position p_{i+1} and orientation ξ_{i+1} of the quadrotor are integrated in a similar manner. Note that the Euler angles ξ wrap around 2π radians which is incorporated into the integration scheme, as well.

The thrust scaling gain k_t is assumed to be constant during integration, i.e. $k_{ti+1} = k_{ti}$. The thrust gain usually changes with the mass of the vehicle and the battery voltage of the vehicle. These effects are not observable from the predicted measurements alone. Learning to predict the change in thrust gain did not provide any improvement in the results during training.

2.5 Estimating hidden state

So far in the network architecture, we predicted the measurements at the next time step given the controls u_i and the state x_i at the current step. Thus, in order to predict sensor measurements for a sequence of time steps $t_{0:N}$, we require the controls along the time steps $u_{0:N-1}$ and the state of the system at the first step x_0.

Since we selected the hidden state as the dynamic state of the system, we can use traditional estimation methods to find the hidden state given the sensor measurements. The position, orientation and joint angles of the hidden state are obtained from the sensor measurements directly. The velocities, Euler angle rates, and joint velocities are obtained by filtering the finite difference differentiated sensor measurements. The commanded Euler angles and joint angles are obtained by integrating the commanded rates and assuming the system starts with commanded angles equal to the measured angles. Finally, the thrust gain k_t is obtained by filtering individual measurements of thrust gain. A single measurement of thrust gain is obtained by inverting (**??**) i.e $k_t = \bar{z}^T(a + \mathbf{g})/a_d$. The accelerations in the equation are obtained by finite differentiating measured velocities and smoothing them with an exponential filter.

2.6 Training

During the training phase, both the feedforward model parameters $\theta = [k_{p_\xi}, k_{d_\xi}, k_{p_r}, k_{d_r}]$ and the weights associated with the fully connected layers are learned together. Since both the fully con-

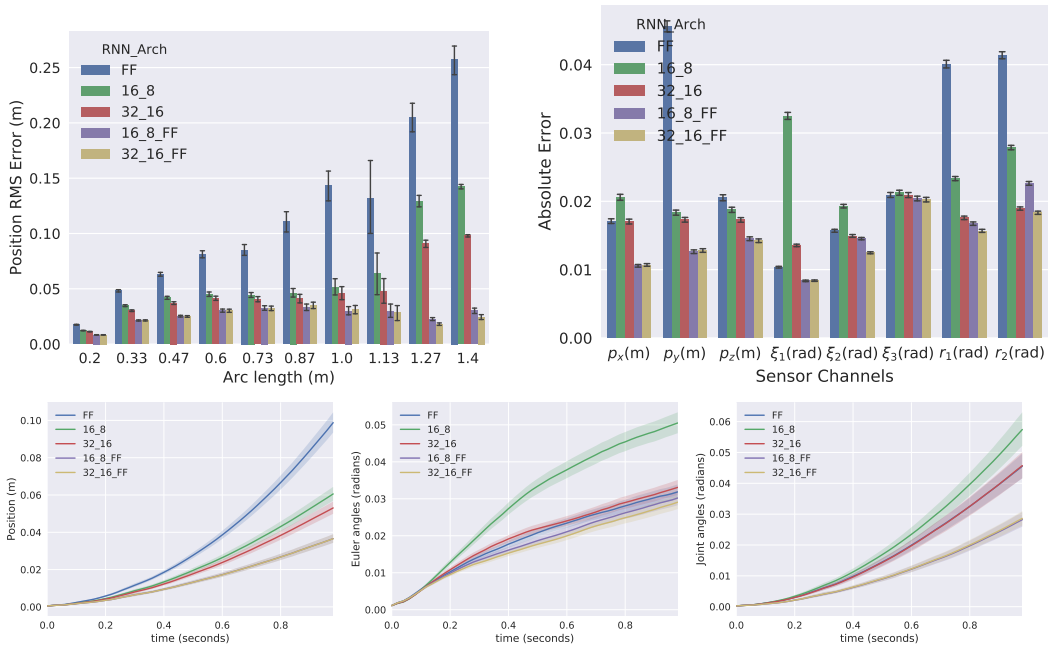


Figure 2: RMS errors along with 95 percent confidence interval on test dataset. (a) Position RMS error against length of the trajectory. Trajectories that have larger arc length are harder to model. (b) RMS Error for each sensor channel. (c) Position RMS error against time (d) Orientation RMS error against time (e) Joint angle RMS Error against time

nected layers and the feedforward model are trying to generate the same quantity, i.e. accelerations for the dynamic system, there will be multiple solutions to the parameters which can produce the same accelerations. Thus, we incorporate a prior on the feedforward model parameters to ensure the model is generalizable. This prior has been obtained by training the model using only the feedforward dynamics without adding the residual dynamics. A dropout layer has also been incorporated on the output of the fully connected layers to ensure the feedforward model is applying the right accelerations even when the residual network output is noisy.

The training data consists of several sequences of human piloted quadrotor data along with sinusoidally oscillated arm data with a variety of frequencies, phases, and offsets. The data has been verified to cover as much of the state space as possible keeping in mind the safety of the vehicle. These sequences are further split into several smaller segments with a fixed horizon length of 1 second with a back propagation length of N segments (In our case we used $N = 50$ segments since the motion capture data is sampled at 50 Hz). The state of the system is estimated along the entire sequence of collected trajectories and the starting estimated state for each sequence has been stored for training purposes. Finally, stochastic gradient descent has been applied on the cost function shown in (**??**) where the predicted sensor measurements are obtained by unrolling the neural network using the applied controls and initial sensor measurement for the sequence. The cost function consists of the error between predicted sensor measurements $z_{1:N}$ and the observed sensor measurements $\bar{z}_{1:N}$ in addition to regularization costs for the feedforward parameters θ.

$$L = \sum_{i=0}^{Nb} \frac{1}{N} \sum_{j=q}^{N} (z_j - \bar{z}_j)^T \Sigma_z^{-1} (z_j - \bar{z}_j) + \frac{1}{2}(\theta - \theta_p)^T \Sigma_\theta^{-1} (\theta - \theta_p), \tag{9}$$

where Σ_z is the covariance in sensor measurements, θ_p is the prior on the feedforward model parameters, Σ_θ is the covariance associated with the prior.

2.7 Prediction Results

The model obtained from training has been verified on a test data set. Fig. 2 shows the position RMS errors against the length of the trajectory (arc length). The feedforward model predicts smaller length

trajectories better and its performance degrades with the length of the trajectory. This behavior is also exhibited by neural networks without including the feedforward network. The performance of the RNN augmented by feedforward model does not degrade with arc length and is almost constant.

The RMS error along each of the sensor channels is shown in Fig. 2. It can be observed that without a feedforward model, the smaller RNN network shown in green has a large Euler angle error compared to a pure feedforward model. Using a larger RNN network overcomes this issue but requires more computational effort which increases the optimization time during NMPC. The RNN model augmented with feedforward dynamics improves only slightly when the number of nodes in the network are doubled. This suggests that using the smaller RNN network combined with a feedforward model is sufficient for NMPC control.

The RMS errors of the sensor channels against the time horizon are shown in Fig. 2. The position errors increase nonlinearly with time since the position of the UAV is predicted based on accurate prediction of other states such as velocities and accelerations. The Euler angle prediction error increases linearly for most of the RNN and feedforward models. The joint angles on the other hand saturate to an RMS value that decreases with an increase in RNN prediction capacity. The RMS errors for all the sensor channels start out with a very small value and degrade gradually. This is a necessary property for the model to be useful in an NMPC scheme.

3 Nonlinear Model Predictive Control

NMPC control solves an optimization problem at each control step to find a control that can track a user-defined reference trajectory while also satisfying control bounds and minimizing a cost function along the trajectory. In this work, we use a quadratic cost that penalizes the deviation from a reference trajectory $x_{d0:N}$ and forces the control effort $u_{0:N-1}$ to be close to the reference controls $u_{d0:N-1}$. The gain matrices Q, R used in the cost function are chosen to normalize the error based on the allowed tolerances for each dimension along the state and control vectors. The normalization of the cost function avoids biasing towards dimensions with larger errors. The cost can be written as

$$L = \sum_{i=0}^{N} (x_i - x_{di})^T Q (x_i - x_{di}) + \sum_{i=0}^{N-1} (u_i - u_{di})^T R (u_i - u_{di}), \tag{10}$$

$$\text{s.t } x_{i+1} = f(x_i, u_i, \theta), \;\; u_{lb} \leq u_i \leq u_{ub}, \tag{11}$$

where f encodes the dynamics and u_{lb} and u_{ub} are the lower and upper bounds on the controls, respectively. We employed a Stage-wise Newton method [28] to solve the trajectory optimization problem specified in (**??**). Stage-wise Newton is a second order method that performs a backward pass and forward pass during each iteration. During the backward pass, the optimal control update directions are obtained by minimizing a second order approximation of the value function around the current solution. In the forward pass, the control perturbations are added to the existing control solution and the state trajectory is updated using the dynamics. The step size for perturbing the control vector is searched over by unrolling the dynamics using the perturbed control vector and checking if the cost decreases in the forward pass. This optimization is very efficient and usually takes only a couple of iterations to find the optimal solution.

Since the NMPC scheme performs a new optimization at every control step, it is necessary to keep the optimization times small. The optimization process is usually expensive without a good initial guess for the controls. Assuming the quadrotor dynamics is smooth, the optimization problem solved at each step is very close to the one from the previous step. Hence, we initialize the optimization problem by shifting the controls obtained from the previous optimization while extending the last control value to fill the rest of the trajectory, i.e. the new initial guess is given by

$$u_{0:N-1} = [u_k, u_{k+1}, u_{N-1}, \underbrace{\cdots}_{\text{(N-k) times}}, u_{N-1}], \tag{12}$$

where k is chosen based on the elapsed time between optimizations.

4 Results

The NMPC optimization has been tested on a DJI Matrice UAV equipped with a 2DOF arm to track spiral reference trajectories for the UAV and sinusoidal reference trajectories for the arm. The

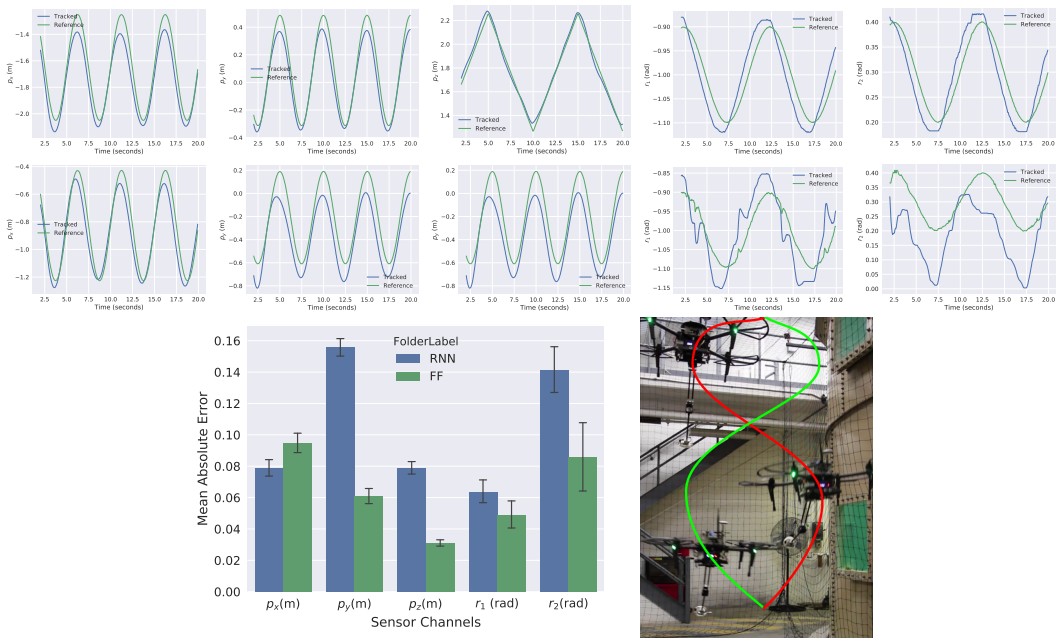


Figure 3: MPC tracking errors; Top row shows tracking errors for position and joint angles using the pure feedforward model. Second row shows the same tracking using the RNN model. The third row (left) shows the absolute errors and the 95% confidence interval around the mean absolute error for each of the error channels and (right) a photo of the aerial manipulator tracking a spiral trajectory.

NMPC optimization runs at a frequency of 50 Hz onboard a core-i5 NUC mini-PC. The UAV pose is obtained from motion capture cameras and joint angles are obtained from servo feedback.

Figure 3 shows the trajectory followed by the UAV, joint angles, and the reference trajectories commanded for the RNN model and the feedforward model. The mean absolute errors for each position axis are shown in the bottom plot. As can be observed, the RNN and feedforward model perform equally well in tracking the UAV trajectory. The RNN model slightly underperforms in tracking joint angles. A possible reason for this is that the feedforward model is very good at predicting the joint angle trajectories, and the addition of a neural network is adding noise to the model optimization. Another possible concern is that the controls being optimized over are not constrained to be from the same distribution as the training data sets. Hence the NMPC optimization can find controls that minimize the cost function but the controls might not be physically meaningful. In-spite of these issues, the RNN model performed well in tracking spiral reference trajectories with an approximate position accuracy of 0.1 meters and joint angle accuracy of 0.1 radians as shown in Fig. 3.

5 Conclusions

This work showed the use of an RNN model to predict the coupled dynamics of an aerial manipulator. Combining the RNN model with a feedforward model encoding the known dynamics of the system improved the prediction performance with a small increase in the number of parameters. We employed the RNN model and the feedforward model in an NMPC framework to track reference trajectories for the arm and the UAV. The RNN model performed comparable to the feedforward model and did not provide any notable improvements in tracking performance. Future work will learn to model physical interactions with the environment and applying the NMPC framework to handle external disturbances safely.

Acknowledgments

This work is supported by []. The authors would also like to thank [] for his help in conducting the hardware experiments and all the reviewers who provided very useful comments.

References

[1] Amazon. Prime air. https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011, 2017.

[2] C. Huerzeler, G. Caprari, E. Zwicker, and L. Marconi. Applying aerial robotics for inspections of power and petrochemical facilities. In *2012 2nd International Conference on Applied Robotics for the Power Industry (CARPI)*, pages 167–172, Sept 2012. doi:10.1109/CARPI.2012.6473371.

[3] A. Jimenez-Cano, J. Braga, G. Heredia, and A. Ollero. Aerial manipulator for structure inspection by contact from the underside. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 1879–1884. IEEE, 2015.

[4] A. Ichikawa, Y. Abe, T. Ikeda, K. Ohara, J. Kishikawa, S. Ashizawa, T. Oomichi, A. Okino, and T. Fukuda. Uav with manipulator for bridge inspection; hammering system for mounting to uav. In *2017 IEEE/SICE International Symposium on System Integration (SII)*, pages 775–780, Dec 2017. doi:10.1109/SII.2017.8279316.

[5] N. Michael, J. Fink, and V. Kumar. Cooperative manipulation and transportation with aerial robots. *Autonomous Robots*, 30(1):73–86, jan 2011. ISSN 1573-7527. doi:10.1007/s10514-010-9205-0. URL https://doi.org/10.1007/s10514-010-9205-0.

[6] B. Galea, E. Kia, N. Aird, and P. G. Kry. Stippling with aerial robots. In *Proceedings of the Joint Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering*, pages 125–134. Eurographics Association, 2016.

[7] J. Molina and S. Hirai. Pruning tree-branches close to electrical power lines using a skew-gripper and a multirotor helicopter. In *Advanced Intelligent Mechatronics (AIM), 2017 IEEE International Conference on*, pages 1123–1128. IEEE, 2017.

[8] J.-P. Ore, S. Elbaum, A. Burgin, and C. Detweiler. Autonomous aerial water sampling. *Journal of Field Robotics*, 32(8):1095–1113, 2015.

[9] M. Kobilarov. Nonlinear trajectory control of multi-body aerial manipulators. *Journal of Intelligent & Robotic Systems*, 73(1-4):679–692, 2014.

[10] A. E. Jimenez-Cano, J. Martin, G. Heredia, A. Ollero, and R. Cano. Control of an aerial robot with multi-link arm for assembly tasks. In *2013 IEEE International Conference on Robotics and Automation*, pages 4916–4921, May 2013. doi:10.1109/ICRA.2013.6631279.

[11] G. Heredia, A. E. Jimenez-Cano, I. Sanchez, D. Llorente, V. Vega, J. Braga, J. A. Acosta, and A. Ollero. Control of a multirotor outdoor aerial manipulator. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3417–3422, Sept 2014. doi:10.1109/IROS.2014.6943038.

[12] Dji a3 autopilot, 2018. URL https://www.dji.com/a3.

[13] B. Erginer and E. Altug. Modeling and pd control of a quadrotor vtol vehicle. In *2007 IEEE Intelligent Vehicles Symposium*, pages 894–899, June 2007.

[14] M. Orsag, C. Korpela, and P. Oh. Modeling and control of mm-uav: Mobile manipulating unmanned aerial vehicle. *Journal of Intelligent & Robotic Systems*, 69(1):227–240, Jan 2013. ISSN 1573-0409. doi:10.1007/s10846-012-9723-4. URL https://doi.org/10.1007/s10846-012-9723-4.

[15] K. Kondak, F. Huber, M. Schwarzbach, M. Laiacker, D. Sommer, M. Bejar, and A. Ollero. Aerial manipulation robot composed of an autonomous helicopter and a 7 degrees of freedom industrial manipulator. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2107–2112, May 2014.

[16] J. Fink, N. Michael, S. Kim, and V. Kumar. Planning and control for cooperative manipulation and transportation with aerial robots. *The International Journal of Robotics Research*, 30(3):324–334, 2011.

[17] K. Kondak, A. Ollero, I. Maza, K. Krieger, A. Albu-Schaeffer, M. Schwarzbach, and M. Laiacker. *Unmanned Aerial Systems Physically Interacting with the Environment: Load Transportation, Deployment, and Aerial Manipulation*, pages 2755–2785. Springer Netherlands, Dordrecht, 2015. doi:10.1007/978-90-481-9707-1_77. URL https://doi.org/10.1007/978-90-481-9707-1_77.

[18] H.-N. Nguyen, S. Park, J. Park, and D. Lee. A novel robotic platform for aerial manipulation using quadrotors as rotating thrust generators. *IEEE Transactions on Robotics*, 34(2):353–369, 2018.

[19] A. Nikou, C. Verginis, S. Heshmati-alamdari, and D. V. Dimarogonas. A nonlinear model predictive control scheme for cooperative manipulation with singularity and collision avoidance. In *2017 25th Mediterranean Conference on Control and Automation (MED)*, pages 707–712, July 2017.

[20] M. Neunert, C. de Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli. Fast nonlinear model predictive control for unified trajectory optimization and tracking. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1398–1404, May 2016. doi:10.1109/ICRA.2016.7487274.

[21] D. Lunni, A. Santamaria-Navarro, R. Rossi, P. Rocco, L. Bascetta, and J. Andrade-Cetto. Nonlinear model predictive control for aerial manipulation. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 87–93, June 2017.

[22] A. Carrio, C. Sampedro, A. Rodriguez-Ramos, and P. Campoy. A review of deep learning methods and applications for unmanned aerial vehicles. *Journal of Sensors*, 2017, 2017.

[23] B. Maciel-Pearson and T. Breckon. An optimised deep neural network approach for forest trail navigation for uav operation within the forest canopy. 2017.

[24] J. Heylen, S. Iven, B. De Brabandere, O. Mogrovejo, J. Antonio, L. Van Gool, and T. Tuytelaars. From pixels to actions: Learning to drive a car with deep neural networks. In *IEEE Winter Conference on Applications in Computer Vision*, 2018.

[25] K. Kelchtermans and T. Tuytelaars. How hard is it to cross the room?–training (recurrent) neural networks to steer a uav. *arXiv preprint arXiv:1702.07600*, 2017.

[26] I. Lenz, R. A. Knepper, and A. Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*, 2015.

[27] G. Garimella, J. Funke, C. Wang, and M. Kobilarov. Neural network modeling for steering control of an autonomous vehicle. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2609–2615, Sept 2017. doi:10.1109/IROS.2017.8206084.

[28] D. P. Bertsekas. *Nonlinear Programming, 2nd ed.* Athena Scientific, Belmont, MA, 2003.