Perception-Based UAV Fruit Grasping Using Sub-Task Imitation Learning

Gabriel Baraban[1], Siddharth Kothiyal[2] and Marin Kobilarov[1]

***Abstract*— This work considers autonomous fruit picking using an aerial grasping robot by tightly integrating vision-based perception and control within a learning framework. The architecture employs a convolutional neural network (CNN) to encode images and vehicle state information. This encoding is passed into a sub-task classifier and associated reference waypoint generator. The classifier is trained to predict the current phase of the task being executed: *Staging*, *Picking*, or *Reset*. Based on the predicted phase, the waypoint generator predicts a set of obstacle-free 6-DOF waypoints, which serve as a reference trajectory for model-predictive control (MPC). By iteratively generating and following these trajectories, the aerial manipulator safely approaches a mock-up goal fruit and removes it from the tree. The proposed approach is validated in 29 flight tests, through a comparison to a conventional baseline approach, and an ablation study on its key features. Overall, the approach achieved comparable success rates to the conventional approach, while reaching the goal faster.**

I. Introduction

Unmanned aerial vehicles (UAV) are well suited to carry out tasks which would be otherwise inefficient or unsafe for human-piloted vehicles. These devices have been utilized successfully in autonomous racing, photography, and mapping. An unmanned aerial vehicle (UAV) must be aware of its environment to avoid collisions en route to its specified goal. In aerial manipulation tasks, such as package delivery [1], infrastructure inspection [2], or construction [3], the UAV must interact with the environment, not merely avoid it. This requires two important processes: sensing and path planning. In the first, information from onboard sensors (cameras, LIDARs, IMUs, etc.) is used to map out and understand the UAV's surroundings, estimating objects of interests such as goals and the obstacles. In the second, this information about the environment is used to generate a plan of action, allowing the UAV to achieve its aim safely and efficiently.

This work describes the development of an algorithm for picking fruits off of trees using a system comprised of a quadrotor with an attached rigid arm. This system is under-actuated and has no prior knowledge of the environment. A convolutional neural network (CNN) is taught to mimic an expert trajectory planner, and to predict a safe, collision-free set of waypoints that converge towards the desired goal (i.e. an orange). The ResNet-based CNN is trained end-to-end to create a deep integrated pipeline for both sensing and path planning. This allows the model to integrate visual features largely ignored by prior methods, such as leaf edges, corners, and shadows, when inferring waypoints to guide the UAV. The model outputs waypoints to guide the UAV. Dynamic feasibility is enforced by an optimization step using Differential Dynamic Programming (DDP) to produce a dense trajectory out of these inferred waypoints. By using this network as a receding horizon path planner, the UAV is able to bundle together successive predictions and reach the goal.

Imitation learning has been trained to successfully perform complex tasks with robotic systems. Conventional imitation learning pipelines generally require a large amount of data, as the dataset must cover most of the spectrum of the expected input data space. Dataset creation for such pipelines also relies on the data being representative of all scenarios encountered by the robotic system. Without this equal distribution, imitation learning based systems can suffer in quality, requiring supplemental strategies such as DAgger [4] to improve the model performance (as in Loquercio et al. [5]).

This work proposes a variation to the conventional imitation learning, as shown in Figure 3. The network contains a common ResNet like CNN sub-network, followed by a set of dense sub-networks, each used to perform waypoint prediction during different phases encountered by the UAV. Sharing a common CNN allows the network to encode the visual information sensed from the environment into a latent vector representation, which is then used by each of the dense sub-networks for optimizing waypoint prediction in their respective phases. The latent vector is concatenated with state information before passing through the dense sub-networks. The network also takes in as input state information, which is concatenated to the latent vector representation encoded by the CNN.

In order to generate a dataset for training the network, a visual-servoing method was created. It solves the fruit-picking problem using a state machine implemented in the Aerial Autonomy [6] software architecture. This state machine acts as a deterministic Markov Decision Process (MDP), with each state solving a distinct sub-problem of the overall fruit-picking task. This method was also used as a comparison baseline for the network-based solution.

A. Problem Formulation

The system used in this work is an aerial manipulator with a fixed arm. The position of the quadrotor base, $p \in \mathbb{R}^3$, its orientation $R \in SO(3)$, and the arm joint angles $r \in \mathbb{R}^2$ together form the *posture* of the system, $q = (p, R, r)$. This

[1]Gabriel Baraban and Marin Kobilarov are with the Department of Mechanical Engineering, Johns Hopkins University, 3400 N Charles Str, Baltimore, MD 21218, USA gbaraban|marin@jhu.edu

[2] Siddharth Kothiyal is with the Laboratory for Computational Sensing and Robotics, Johns Hopkins University, 3400 N Charles Str, Baltimore, MD 21218, USA sidkothiyal@jhu.edu

defines two transforms in SE(3): $g(q) = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}$, the location and attitude of the quadrotor base, and $\Phi(r)$ the relative pose of the end effector in the body frame. When multiplied, the resulting transform $g_e(q) = g(q)\Phi(r) \in SE(3)$ is the position and orientation of the end effector in world coordinates. The geometry of the system in euclidean space is defined as $\mathcal{A}(q)$, the set of points within the workspace $\mathcal{W}$.

The system velocity $\dot{q} = (v, \omega, \dot{r})$, is the derivative of each element of q: the body-fixed linear and angular velocities $v, \omega \in \mathbb{R}^3$ and the joint velocities $\dot{r} \in \mathbb{R}^2$. Because the arm is fixed, $\dot{r} = 0$. The velocities are used in homogeneous coordinates using the formulas:

$$\widehat{V} = \begin{bmatrix} \widehat{\omega} & v \\ 0 & 0 \end{bmatrix}, \quad \widehat{\omega} = \begin{bmatrix} 0 & -\omega_x & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}.$$

The full state of the system $x = (q, \dot{q})$ is controlled by the input $u \in \mathbb{R}^4$. u is composed of three torques around the body-fixed axes, and a thrust force applied along the local z-axis. The dynamics are of the form $\dot{x} = f(x, u)$ based on the derivations in [7] and [8]:

$$\dot{g}(q) = g(q)\widehat{V}, \quad M(q)\dot{V} + b(x) = B(q)u \tag{1}$$

$M(q), b(x), B(q)$ are the mass matrix, bias vector, and control transformation matrix.

The goal of robotic fruit picking is to compute a reference trajectory $\bar{x}(t)$ and a series of control inputs $u(t)$ minimizing the cost:

$$\int_{t_0}^{t_f} \frac{1}{2}\|x(t) - \bar{x}(t)\|_Q^2 + \frac{1}{2}\|u(t)\|_R^2 dt$$

subject to the constraints:

$$\dot{x} = f(x(t), u(t)) \tag{2}$$

$$\mathcal{A}(q(t)) \cap E = \emptyset \tag{3}$$

$$g(\bar{x}(t_f))\Phi(r) = \begin{bmatrix} \star & p_{goal} \\ 0 & 1 \end{bmatrix} \tag{4}$$

The first constraint (Equation 2) enforces the dynamics from equation 1. The second constraint (Equation 3) prevents the system from colliding with the environmental obstacles, defined through the set of points $E \subset \mathcal{W}$. The third constraint (Equation 4) requires that the final posture of the system place the end-effector position at $p_{goal} \in \mathbb{R}^3$, the location of the fruit. The values of x are fully observable, and accessed through the hardware suite onboard the system. E and p_{goal} are not known *a priori* and must be estimated online through a fusion of camera images and state measurements.

B. Imitation Learning Sub-Tasks

The fruit-picking task outlined above requires a wide variety of actions from the UAV. At the beginning of a trial, it moves quickly to close the gap between the vehicle and the tree. Once it nears the tree, its speed slows. It must move more precisely, avoiding collisions with the tree and lining up the end-effector with the fruit. This, in addition to the dynamical relationship between successive states, prevents the training dataset from being independent and identically distributed (i.i.d), a common expectation of machine learning data.

To allow for visual-servoing, the larger task of fruit picking was modelled as an MDP, divided into three control phases. Each phase solves a discrete-time stochastic control problem:

- *Staging:* The quadcopter begins in this state and makes large motions to reach an offset location within 1 meter of the goal.
- *Picking:* Once the goal is close, the quadcopter takes smaller, more-precise steps to grasp the fruit.
- *Reset:* If any malfunction occurs during the *Picking* phase, the system returns to the *Staging* location, so it can make another attempt.

Each of these phases has greater internal consistency, and training a separate model for each phase leads to better learning performance than training one network for the entire dataset. Training these networks is made more efficient by using a shared visual CNN-based encoder, as a significant amount of visual information will have an overlap in multiple phases. In addition to learning behaviors for each phase, the network also learns to predict its current phase, thereby choosing the next action to follow.

C. Related Work

1) Aerial Manipulation

Aerial manipulation research varies widely, using systems of a variety of sizes, sensors, and end-effectors. Some works, such as Zhang et al. [9], use overactuated systems, allowing the end-effector dynamics to be completely decoupled from the UAV. Others prefer lightweight underactuated systems, using the coupling between the UAV and end-effector as an advantage. For example, Welde et al. [10] prove that underactuated aerial manipulators are differentially flat, allowing for easy motion planning in the flat output space and Spurny et al. [11] demonstrate an algorithm for cooperative search, picking, and placing of a set of unknown ferrous objects using a team of small UAVs. For sensing, a aerial manipulators often use onboard cameras, allowing for visual servoing, such as in Mebarki et al. [12]. This visual information can be synthesized with IMU data to perform visual-inertial odometry, as in [13]. Other works choose less conventional sensing techniques, such as Mulgaonkar et al. [14], which uses collisions with the environment to create a map of the surroundings. The manipulators used in different research works range from simple 1-DOF grippers [11], to more complex open-chain manipulators [9], to bimanual end-effectors for more complex manipulations [15]. More detail on the diversity of aerial manipulation systems used in research can be found in the literature reviews by Ding et al. [16] and Samadikhoshkho et al. [17].

2) Mapping

Classical machine learning approaches to UAV navigation aim to distill an image into specific measurements of object locations and orientations, which are then passed into a planning algorithm. UAV control algorithms for obstacle avoidance often use an explicit map of the environment. This may be accomplished through a point cloud (often provided by a depth camera) with processing via algorithms like RANSAC [18] or point clustering [19]. In another

application,Francis et al. [20], use the differences between successive frames to model the motion of pixels in the scene.

Beyond these classical methods, Foehn et al. [21] use convolutional neural networks (CNNs) to map the locations of objects in an environment (in their work, gates for drone racing). Liu et al. [22] uses CNNs to map the location of fruit in mango orchards. The above methods remove all visual information not related to this explicit output.

3) Path Planning

Given input information such as a map or a measurement of goal and obstacle locations, there are several ways to produce a safe, feasible trajectory. Typically, these approaches fall into two categories, sampling and optimization. Sampling approaches search the state space [23] or control space [24], discarding any values that result in unsafe behaviour. These safe states are then converted into a full trajectory, through a minimum snap trajectory [25], [26], a polynomial or b-spline fit [27]. Optimization approaches focus exclusively on identifying and minimizing costs, skipping the sampling step. They apply costs to collisions and rewards to reaching goal spaces, (see [28]). Other methods, such as [29], [30], find time-optimal paths.

Machine learning approaches to path planning typically take in the state and environment observation vectors and use fully connected networks to infer control actions. Hwangbo et al. [31] use reinforcement learning to train a UAV to quickly reach a stable equilibrium from any random initialization. Song et al. [32] use a similar approach, but use an observation of racing gates to infer u as well.

4) Imitation Learning

The present work aims to explore UAV navigation in uncertain environments. In such contexts, Kaufmann et al. [33], use deep learning to infer commands directly from images, allowing the UAV to perform acrobatic maneuvers without full state information. Similarly, the NVIDIA PilotNet [34] infers a ground vehicle steering angle,and Yang et al. [35] builds on PilotNet to output both a steering angle and speed goal. The above works train networks using a regression-based cost function. The cost function used in training the fruit-picking network is based on work by Kim et al. [36], which found that classification-based cross-entropy loss can improve on the traditional regression-based loss.

II. Baseline Approach

Before training a network to pick an fruit, we first develop a method using existing techniques. This allows the novel approach below to be compared with the current state of the art. In this baseline approach a neural network segments the fruit from the color image. The centroid pixel is then back-projected using the depth image, providing a three dimensional fruit position in the camera frame. Finally, the pixels around the fruit are used to estimate a non-colliding approach angle.

A. Control Framework

The baseline control algorithm performs the following steps (shown in Figure 1):

1) The onboard camera captures color and depth images.
2) The color image is passed through a segmentation network to find the location of the goal.
3) The location of the centroid in the camera coordinate frame is calculated using the depth image and the camera's intrinsic matrix.
4) The points near the centroid are back-projected, creating an estimate of the local tangent plane.
5) A goal position and yaw are calculated from the position of the fruit and the normal vector to the tangent plane.
6) A polynomial reference trajectory is calculated, leveraging the differential flatness of the quadcopter dynamics to generate a trajectory $\bar{x}(t)$ from the position and yaw goal.
7) A low-level controller tracks $\bar{x}(t)$ while a new trajectory update can be produced from the next camera frame.

B. Segmentation

To be able to successfully attempt trajectory generation to the goal, the environment must be sensed to locate the fruit. Having access to only the onboard camera for sensing, a custom segmentation network (based on the U-Net architecture [37]) was trained to achieve this task. For segmenting out the fruit in the image, four convolutional layers, of sizes 128, 256, 512, and 512, respectively, are applied sequentially to the image. Three transpose convolutional layers, of sizes 512, 256, and 128, are then applied, creating an output of the same size as the original input image. Two skip connections are added: one between layer 1 and layer 7, and the other between layer 2 and layer 6. After training on a dataset of 10,000 hand-labeled images, the network was able to predict the location of the fruit in the picture with an Intersection over Union of 93.49%. The result can be seen in Figure 2.

C. 3D Projection and Plane Estimation

The centroid of the segmented pixels are then projected to 3D using the depth channel of the camera and the pinhole camera projection model. Knowing the location of the fruit gives the algorithm the information it needs to reach the final goal, but not enough to avoid collisions with the environment. In order to approach the fruit from a safe angle, the pixels near the centroid are back-projected into three dimensional space and transformed into world frame coordinates. The resulting point cloud is fit to a plane using RANSAC [18]. The normal vector to this plane η is an estimate of the safest approach angle for the aerial manipulator. In order to ensure safe flight, the z coordinate of η is set to zero, so that the quadcopter will approach from a horizontal direction. An example is shown in Figure 2.

D. Trajectory Generation

During flight, the segmentation and plane estimation processes occur at 5 hz, continuously updating the estimate of p_{goal} and η. As this system completely depends on the camera for estimating the location of the fruit, it must ensure the fruit remains in view throughout the process. To ensure this happens smoothly, especially towards the end of the process (where the gripper is likely to block the view to the fruit), a staging state is added, which makes the quadcopter

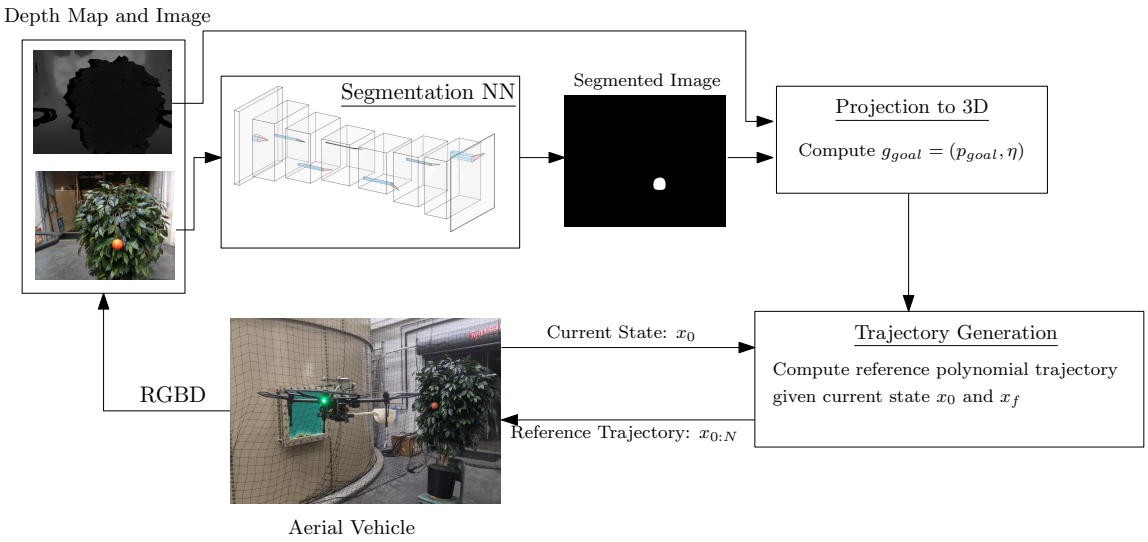

Fig. 1. Flow of data for baseline approach

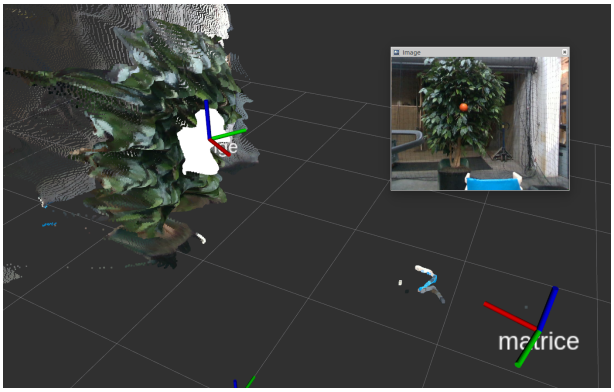

Fig. 2. The point-cloud generated by the onboard camera. The points used for plane estimation are shown in white, with the estimate of η shown as the red axis at the centroid.

position itself at an offset slightly below and in front of the fruit. At the beginning of each flight, the MDP phase is initialized to *Staging*. The algorithm computes a safe location at the predefined offset from p_{goal} in the direction of η. A polynomial reference trajectory in position and yaw is computed to bring the quadrotor from its current location to the staging location. The other elements of the state (roll, pitch, v, and ω) are calculated from the position and yaw using the differential flatness of the system. After reaching the goal pose, the MDP transitions into the *Picking* phase. Another polynomial trajectory is calculated, this time to bring the end effector to p_{goal}. Moving to the offset location causes the final approach angle to p_{goal} to be parallel to η, and therefore less likely to collide with the tree. This phase ends when the end effector detects a successful connection with the fruit. If the *Picking* phase takes too long without reaching the fruit or the fruit goes out of view, the *Reset* stage begins, which returns to the same offset location as the end of *Staging* and then restarts *Picking*.

E. Shortcomings

This approach is effective only when the fruit is in view during flight. If the fruit is not initially in view, the approach has no means to explore and find it. Relatedly, if the fruit leaves view during the *Staging* phase, the system has no recourse but to backtrack and try again. If the fruit leaves view during the *Picking* phase, it has the ability to retry *Picking*, but this leads to slower behavior. Second, the segmentation network accuracy is reduced when the fruit is visible but occluded by leaves.

Another limitation is that the RANSAC step is sensitive to outliers in the point cloud. If the fruit is only sparsely surrounded by leaves, the point cloud can include pieces of the background or trunk, making the estimate of η unreliable. Some of this can be effectively mitigated by low-pass filtering. However, when the outliers are dependent on the camera perspective, η can oscillate between estimates, making staging impossible.

Finally, the biggest shortcoming of this method is the rigid logic governing the phase transitions. Often, during the *Staging* phase, the drone will overshoot its target position and retreat to it. This maneuver actually takes it further away from the fruit, wasting time and control effort. The aim of this work is to use a neural network to imitate the successes of this approach, while streamlining operation and avoiding its flaws.

III. Learned Multi-Phase Approach

A. Control Framework

In this work, we use imitation learning via a neural network trained on successful executions of the baseline approach. The procedure follows the following steps:

1) The camera image is received and passed through the segmentation network from section II-B.
2) The color and segmentation layers of the image are fed into a CNN encoder. The resulting latent vector is combined with a vector of the current roll, pitch, and back-projected fruit pose and fed into a densely connected

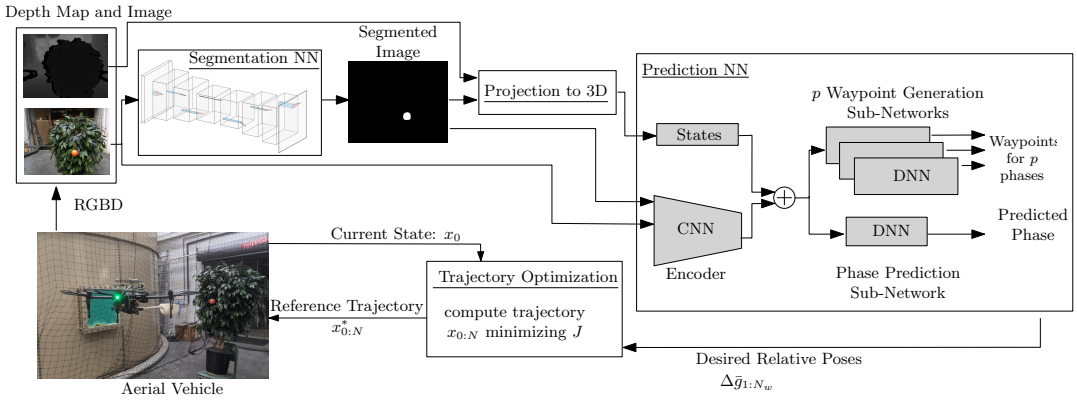

Fig. 3. Flow of data for novel approach

network, trained to output which phase is currently active, as well as $N = 3$ poses $\{\Delta\bar{g}_1, ..., \Delta\bar{g}_N\}$, corresponding to the the desired states over the next N intervals of a time-step Δt.

3) The series of poses $\Delta\bar{g}_i$ are given to a DDP optimizer which produces a smooth trajectory. This trajectory passes through the $\Delta\bar{g}$ waypoints while minimizing control effort and obeying the dynamics of the system.
4) A low-level controller tracks the smooth trajectory while a new trajectory can be produced from the next camera frame.

B. Network Architecture

The waypoint prediction network architecture (shown in Figure 3) begins with a four channel image input, the RGB image from the camera concatenated with the segmented image Section II-B. This visual information, a 640 x 480 x 4 tensor, is passed into a 56 channel CNN and then through three ResNet blocks. Each ResNet block performs the following transformation $y(x) = C_i(Re(C_i(Re(x)))) + C_i(x)$, where Re is the ReLu activation function and C_i is a convolutional layer with i channels (the blocks use $i = 72$, $i = 96$ and $i = 128$ channels respectively). The output after the third block is flattened into a vector, which represents the latent state information encoded from the image data.

This vector is concatenated with the current roll, pitch, and relative orange pose and passed into two fully connected layers of size 16,384 and 8,192. The final step is an output layer of size $(p + p \times N \times b \times 6)$, where $p = 3$ is the number of phases, $b = 100$ is the number of bins used for the classification cost (discussed in Section III-B.1), $N = 3$ is the number of waypoints being generated by the network, each with 6 DOF: x,y,z,yaw, pitch,roll.

The first waypoint $\Delta\bar{g}_0 \in SE(3)$ defines the transform from the current position to a new goal position to be reached after Δt seconds. The subsequent outputs $\Delta\bar{g}_{i=1,2}$ are relative to the $i-1$ transform. By using this change of coordinates, the network can learn to output $\Delta\bar{g}_i = \vec{0}$ when the UAV should move to $\Delta\bar{g}_{i-1}$ and stop.

1) Classification Cost

In training the $\Delta\bar{g}$ outputs, we use a classification cost, similar to that used by Kim et al. [36]. The available space of outputs is discretized into b sections ("*bins*") in each degree of freedom, and the network is trained to output the likelihood that the correct waypoint falls within each bin. This avoids the common pitfall in regression training of unintentionally training a network with low variance which simply outputs the dataset mean. Instead, the probability output is compared to the one-hot output of the expert. During training, the loss function is the cross-entropy between these two distributions. Two concerns when designing this approach are the inherent limitation of the output space, as each coordinate can now only be inferred to fall within the space spanned by the bins, and the network output size, as it is much larger than a direct regression would require. The output space limitation is mitigated by the phase selection architecture, as each MDP phase can be discretized into its own output space. Thus, when in *Picking*, the network will inherently output smaller, less aggressive waypoints than when in *Staging*.

2) Phase Selection

The first $p = 3$ elements of the output vector are treated as a classifier, trained to predict which phase best fits the current system conditions. The rest of the output vector is treated as a series of p distinct outputs, one for each phase. Each output has N points, each with 6 degrees of freedom. Each degree of freedom is encoded as b probabilities, trained through the classification cost. The loss for training the network is computed as the sum of the cross-entropy loss for predicting the phase and the classification cost for predicting the waypoints for the actual phase of the system . During inference, the classifier output is used to determine which set of waypoints will be forwarded along to the trajectory generation algorithm (section III-C), as well as the value of Δt to use. *Staging* behaviors use $\Delta t = 1.0$s, while *Picking* and *Reset* use $\Delta t = 0.25$s.

C. Trajectory Generation

Once the waypoints have been produced, they are passed to a short-term trajectory generator, which uses DDP to produce a trajectory over the next $N \times \Delta t$ seconds that minimizes the cost

$$J = \int_0^{N\times\Delta t} (x(t)^T Q x(t) + u(t)^T R u(t) + w(x,t))\mathrm{d}t$$

The Q cost on $x(t)$ penalizes unsafely high velocities, the R cost on $u(t)$ penalizes high control effort, and the waypoint cost w is zero when $t \neq \{\Delta t, 2\Delta t, ..N \times \Delta t\}$ and $\Delta x^T Q_f \Delta x$ otherwise. This Q_f cost penalizes Δx, the difference between $x(t)$ and $\bar{x}(t) = g(x(0)) \prod_0^{t/\Delta t} \Delta \bar{x}_i$. Thus, through choices of Q, R, and Q_f, DDP finds a trajectory that flies closely to each waypoint, while conforming to the dynamics of the UAV. The trajectory lasts for $N \times \Delta t$ seconds, far longer than it takes for a new image to be processed and a new trajectory to be recalculated.

D. Data Collection

Using the baseline controller, a dataset of approximately 40,000 images and associated paths were collected over the course of 263 trials. Each trial consisted of placing the fruit, tree, and UAV at randomized positions and orientations within the workspace, then allowing the expert to run until the end-effector contained the fruit. This was verified by a magnetometer in the base of the basket and a small magnet glued to the bottom of the fruit. Any trials containing errors in *Picking* (crashing, losing sight of the fruit, getting trapped in an equilibrium) were still used for training *Staging* and *Reset*, provided they completed successfully. The network was trained using these pairs, as well as horizontally mirrored images with the associated waypoints transformed from left to right as well.

IV. Results

A. Hardware

The quadcopter base used in this work is the DJI Matrice 100. It carries an Intel NUC for onboard computation, a Realsense D435i to collect RGB and Depth images, and a LSM303DLHC magnetometer. The fruit is fitted with a magnet on its underside and the magnetometer is used to detect when an fruit has successfully entered the end-effector. The workspace is limited to the volume covered by a Optitrack motion capture system, allowing for 120Hz odometry feedback.

B. Trial Design

After training the network was able to predict the correct phase with 92.2% accuracy in the training dataset and 87.6% accuracy in the validation dataset.

After training, the network was flown in an identical setup to the data collection above (section III-D). Afterwards, the trials were analysed for the following criteria:

1) Picking Success: The percentage of trials in which, after staging, the end effector reached the goal, placing the fruit in the basket.
2) Staging Success: The percentage of trials in which the UAV reached the staging position, leaving only to approach the goal.
3) Picking Speed: On trials which successfully picked fruit, the average speed for the final phase.
4) Staging Speed: On trials which successfully reached the staging position, the average speed of the staging phase.

Mean values of these metrics for the baseline and network controllers can be found in table IV-B.

TABLE I

Comparison of the Baseline and Learned Approaches

Method	Picking Success	Staging Success	Picking Speed (cm/s)	Staging Speed (cm/s)
Baseline	75.6%	99.2%	3.4	12.5
Learned	70.3%	96.3%	2.7	19.2

TABLE II

Ablation Study

Method	Picking Success	Staging Success	Trials
Learned	70.3%	96.3%	29
Without Roll-Pitch	40%	80%	10
Without Fruit Pose	0%	60%	10
Without Multi-phase	10%	70%	10
Regression Loss	10%	20%	10

C. Ablation Study

The network architecture is further validated by an ablation study on four of its features. The same dataset was used to train a set of new networks, one without the roll and pitch inputs, one without the fruit pose inputs, one without multi-phase setup, and one using a regression cost instead of the classification cost. The results of trials with these four networks are shown in table IV-C.

D. Discussion

Compared to the baseline controller, the novel controller performs slightly worse at both staging and picking success. The network demonstrated two common failure modes. First, the UAV would reach a valid staging location, but then remain there instead of entering the *Picking* phase. Second, the UAV would miss the fruit by a small amount, which could knock the fruit to the outside of the basket. The *Reset* behaviour would be unable to disentangle the end-effector from the tree, requiring manual intervention.

The ablated networks performed as follows:

1) The network trained without roll and pitch inputs was prone to more aggressive maneuvers, which often led to the fruit leaving view or the UAV getting too close to the tree.
2) The network trained without fruit pose inputs was dramatically worse, struggling to even reach the staging position.
3) The network trained as a single phase suffered from the equilibrium failures described above, getting to a position with a good view of the fruit and remaining there instead of moving in to pick it.
4) The network trained with a regression cost was not very responsive to changes in its state. It moved generally forward, frequently crashing into branches or missing the tree outright. It only succeeded at reaching the fruit if the randomized initial position placed it directly in front of the fruit.

V. Conclusion

We have presented an algorithm for generating dynamically-feasible paths which enable an aerial manipulator to navigate to and pick a mock-up fruit from

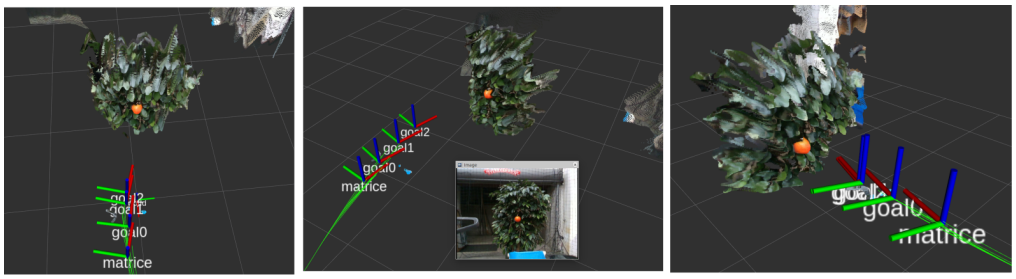

Fig. 4. Three examples of flights using the novel control algorithm. The green lines show the last several computed trajectories. The most recent waypoints, the point cloud, and the onboard camera image are shown as well.

a tree. The main motivation was to enable the system to adapt to complex cases, such as target occlusions, compliant interactions, and scene motion from external disturbances. Since standard geometric model-based planning methods can be sensitive to such uncertainties, the premise was that machine learning could offer a more reliable alternative by more tightly coupling perception and planning.

Admittedly, the advantages of the proposed method as implemented are not obvious from the results obtained up to this point, and further work is needed to investigate its robustness in such complex use cases. The approach, based on imitation learning, is also inherently limited by the quality of training examples. The baseline controller used in this work did not generate perfect performance and, while the network was only trained on its successes, it still inherited some of its flaws. This resulted in the network reproducing behaviors such as getting stuck on foliage. The dataset used to train the network also created a reliance on back-projection of the fruit pose for success, and required near-constant visual contact with the fruit in order to progress.

Looking forward, some of these shortcomings can be alleviated using a larger dataset or adding another sub-network to the architecture. One such improvement could be replacing the plane estimation RANSAC algorithm with an η prediction network. Even more promising are recurrent neural networks, as the internal memory will allow the system to learn the dynamics of the latent space across time steps. An accurate recurrent world model could then serve as a basis for reinforcement learning.

ACKNOWLEDGMENT

Thanks to Professor Joseph Katz and Professor Louis Whitcomb for providing facilities for conducting experiments. This material is based upon work supported by the National Science Foundation under grant no:1527432.

REFERENCES

[1] S. R. R. Singireddy and T. U. Daim, "Technology roadmap: Drone delivery–amazon prime air," in *Infrastructure and Technology Management*, pp. 387–412, Springer, 2018.

[2] A. Caballero, M. Bejar, A. Rodriguez-Castaño, and A. Ollero, "Reactivity and dynamic obstacle avoidance," in *Aerial Robotic Manipulation*, pp. 333–348, Springer, 2019.

[3] F. Augugliaro, S. Lupashin, M. Hamer, C. Male, M. Hehn, M. W. Mueller, J. S. Willmann, F. Gramazio, M. Kohler, and R. D'Andrea, "The flight assembled architecture installation: Cooperative construction with flying machines," *IEEE Control Systems Magazine*, vol. 34, no. 4, pp. 46–64, 2014.

[4] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 627–635, PMLR, 11–13 Apr 2011.

[5] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: From simulation to reality with domain randomization," *IEEE Transactions on Robotics*, vol. 36, no. 1, pp. 1–14, 2020.

[6] M. Kobilarov, G. Garimella, M. Sheckells, S. Kim, and G. Baraban, "Improving the reliability of pick-and-place with aerial vehicles through fault-tolerant software and a custom magnetic end-effector," *IEEE Robotics and Automation Letters*, pp. 1–1, 2021.

[7] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[8] M. Kobilarov, "Discrete optimal control on lie groups and applications to robotic vehicles," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5523–5529, IEEE, 2014.

[9] G. Zhang, Y. He, B. Dai, F. Gu, L. Yang, J. Han, G. Liu, and J. Qi, "Grasp a moving target from the air: System & control of an aerial manipulator," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1681–1687, 2018.

[10] J. Welde, J. Paulos, and V. Kumar, "Dynamically feasible task space planning for underactuated aerial manipulators," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3232–3239, 2021.

[11] V. Spurný, T. Báča, M. Saska, R. Pěnička, T. Krajník, J. Thomas, D. Thakur, G. Loianno, and V. Kumar, "Cooperative autonomous search, grasping, and delivering in a treasure hunt scenario by a team of unmanned aerial vehicles," *Journal of Field Robotics*, vol. 36, no. 1, pp. 125–148, 2019.

[12] R. Mebarki and V. Lippiello, "Image-based control for aerial manipulation," *Asian Journal of Control*, vol. 16, no. 3, pp. 646–656, 2014.

[13] Y. Yang, P. Geneva, K. Eckenhoff, and G. Huang, "Visual-inertial odometry with point and line features," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2447–2454, 2019.

[14] Y. Mulgaonkar, W. Liu, D. Thakur, K. Daniilidis, C. J. Taylor, and V. Kumar, "The tiercel: A novel autonomous micro aerial vehicle that can map the environment by flying into obstacles," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7448–7454, 2020.

[15] A. Suarez, F. Real, V. M. Vega, G. Heredia, A. Rodriguez-Castaño, and A. Ollero, "Compliant bimanual aerial manipulation: Standard and long reach configurations," *IEEE Access*, vol. 8, pp. 88844–88865, 2020.

[16] X. DING, P. GUO, K. XU, and Y. YU, "A review of aerial manipulation of small-scale rotorcraft unmanned robotic systems," *Chinese Journal of Aeronautics*, vol. 32, no. 1, pp. 200–214, 2019.

[17] Z. Samadikhoshkho, S. Ghorbani, F. Janabi-Sharifi, and K. Zareinia, "Nonlinear control of aerial manipulation systems," *Aerospace Science and Technology*, vol. 104, p. 105945, 2020.

[18] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, p. 381–395, June 1981.

[19] B. Xu, W. Li, D. Tzoumanikas, M. Bloesch, A. Davison, and

S. Leutenegger, "Mid-fusion: Octree-based object-level multi-instance dynamic slam," 2019.

[20] S. L. Francis, S. G. Anavatti, and M. Garratt, "Detection of obstacles in the path planning module using differential scene flow technique," in *2015 International Conference on Advanced Mechatronics, Intelligent Manufacture, and Industrial Automation (ICAMIMIA)*, pp. 53–57, 2015.

[21] P. Foehn, D. Brescianini, E. Kaufmann, T. Cieslewski, M. Gehrig, M. Muglikar, and D. Scaramuzza, "Alphapilot: Autonomous drone racing," 2020.

[22] X. Liu, S. W. Chen, C. Liu, S. S. Shivakumar, J. Das, C. J. Taylor, J. Underwood, and V. Kumar, "Monocular camera based fruit counting and mapping with semantic data association," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2296–2303, 2019.

[23] G. Kontoudis, Z. Xu, and K. G. Vamvoudakis, "Online, model-free motion planning in dynamic environments: An intermittent, finite horizon approach with continuous-time q-learning," 07 2020.

[24] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," 2019.

[25] D. Mellinger, Q. Lindsey, M. Shomin, and V. Kumar, "Design, modeling, estimation and control for aerial grasping and manipulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2668–2673, IEEE, 2011.

[26] G. Ryou, E. Tal, and S. Karaman, "Multi-fidelity black-box optimization for time-optimal quadrotor maneuvers," 2020.

[27] D. Jung and P. Tsiotras, *On-line Path Generation for Small Unmanned Aerial Vehicles Using B-Spline Path Templates*.

[28] G. Garimella, M. Sheckells, and M. Kobilarov, "Robust obstacle avoidance for aerial platforms using adaptive model predictive control," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5876–5882, IEEE, 2017.

[29] P. Foehn and D. Scaramuzza, "Cpc: Complementary progress constraints for time-optimal quadrotor trajectories," 2020.

[30] S. Spedicato and G. Notarstefano, "Minimum-time trajectory generation for quadrotors in constrained environments," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 4, pp. 1335–1344, 2018.

[31] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[32] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," 2021.

[33] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Deep drone acrobatics," 2020.

[34] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," 2017.

[35] Z. Yang, Y. Zhang, J. Yu, J. Cai, and J. Luo, "End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perception," 2018.

[36] J. W. Kim, C. He, M. Urias, P. Gehlbach, G. D. Hager, I. Iordachita, and M. Kobilarov, "Autonomously navigating a surgical tool inside the eye by learning from demonstration," 2020.

[37] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.