Advanced Autonomy on a Low-Cost Educational Drone Platform

Luke Eller[1*], Théo Guérin[1*], Baichuan Huang[1*], Garrett Warren[1*], Sophie Yang[1*], Josh Roy[1], Stefanie Tellex[1]

***Abstract*— PiDrone is a quadrotor platform created to accompany an introductory robotics course. Students build an autonomous flying robot from scratch and learn to program it through assignments and projects. Existing educational robots do not have significant autonomous capabilities, such as high-level planning and mapping. We present a hardware and software framework for an autonomous aerial robot, in which all software for autonomy can run onboard the drone, implemented in Python. We present an Unscented Kalman Filter (UKF) for accurate state estimation. Next, we present an implementation of Monte Carlo (MC) Localization and FastSLAM for Simultaneous Localization and Mapping (SLAM). The performance of UKF, localization, and SLAM is tested and compared to ground truth, provided by a motion-capture system. Our evaluation demonstrates that our autonomous educational framework runs quickly and accurately on a Raspberry Pi in Python, making it ideal for use in educational settings.**

I. INTRODUCTION

Substantial increase in demand in the field of robotics demonstrates the need for autonomous educational platforms. The International Data Corporation predicts that global spending on robotic technologies—and drones in particular—will grow annually over the next several years at a compound rate of nearly 20 percent, which is a massive opportunity for continued innovation in the field [1]. However, the plethora of knowledge and technical skills required for this growing domain is a considerable barrier to entry. This paper focuses on making advanced autonomy accessible to individuals with no robotics experience. We build on the low-cost educational platform introduced in [2] by adding advanced algorithms for state estimation, localization, and SLAM. The algorithms are implemented in Python and documented in novel course projects.

Improvement in drone technology has made many commercial autonomous aircraft widely available. These include the Tello EDU [3] from DJI which provides high-level APIs for education and the Skydio R1 [4] which provides an SDK for developing. These commercial drones primarily target high-level programming aspects and are not open-source. There are some open-source drone platforms for advanced college-level courses; however, these are not suitable for students with less background in engineering and robotics [2]. The PiDrone platform and course were created to fill this gap.

*Denotes equal contribution

[1]Department of Computer Science, Brown University, Providence, RI, 02912, USA. Correspondence: {luke_eller, theo_guerin, baichuan_huang, garrett_warren, sophie_yang, josh_roy, stefanie_tellex}@brown.edu

Fig. 1: PiDrone Hardware Platform

For complex tasks such as mapping, precise position control, or trajectory following it is necessary to have precise state estimates of velocity and position. In this paper, we present an onboard UKF to better estimate state, as well as localization and SLAM implementations to generate more accurate position estimates. Mathematical descriptions of these algorithms provide a formal description of the state, observation, and control models used to obtain good performance. Corresponding course material includes these algorithms and was successfully taught to undergraduate students at Brown University in 2018 [5] [6] [7].

In Section IV, we quantitatively evaluate the performance of the UKF and localization running entirely onboard the drone's Raspberry Pi, and FastSLAM running offboard on a separate base-station computer with ROS installed. Even though FastSLAM cannot run onboard in an efficient way, students can still learn and implement the algorithm and then delegate the computation to another machine to increase performance. The accuracy of the position estimates obtained by the UKF, localization, and SLAM compared to ground-truth measured by a motion-capture system exemplifies the drone's ability to serve as an educational platform for these algorithms. The fact that the algorithms can all be written in Python increases the accessibility of these algorithms, which is especially crucial for an introductory robotics course.

II. ARCHITECTURE

A. Hardware

The PiDrone follows a similar parts list to [2] with a few improvements, making it safer and easier to build while remaining under $225. Notable hardware changes include the use of a Raspberry Pi HAT, an add-on board, to improve the accessibility of the build process and the robustness of

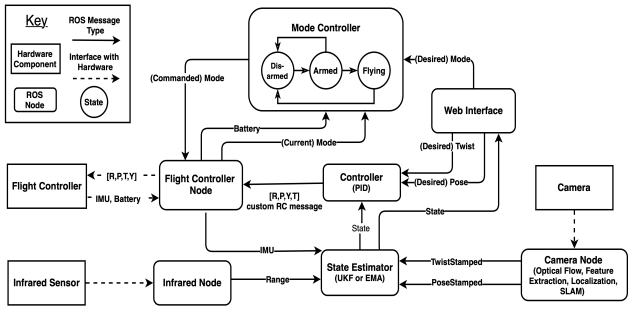


Fig. 2: Software Architecture Diagram

the built drone [8]. Soldering directly to the pins of the Raspberry Pi is a difficult task for introductory students. Instead, the Raspberry Pi HAT allows students to solder to pads rather than pins. We also add lightweight, open-source, 3D printed propeller guards to the platform as seen in Fig. 1. These guards provide additional safety for students and researchers.

B. Software

To maximize ease of use of the PiDrone as an educational and research platform, the software architecture is organized as shown in Fig. 2. The modular architecture provides a plug-and-play software environment for the drone. The ease of swapping core software components including control algorithms, state estimators, sensor interfaces, and user inputs is demonstrated concretely through the corresponding educational course. Students implement and substitute in their own scripts for PID control, SLAM, and UKF state estimation without compromising the stability of the rest of the system. This modularity enables researchers to easily implement and evaluate a new state estimation or control algorithm or add a new sensor to the drone.

III. STATE ESTIMATION

A. Unscented Kalman Filter

The UKF algorithm and the related Extended Kalman Filter (EKF) algorithm are the industry standards for state estimation of nonlinear systems, used by open-source projects such as Cleanflight and Betaflight, as well as commercial products such as the Crazyflie [9] [10] [11]. The PiDrone course covers both the mathematics and use cases of the Kalman Filter, EKF, and UKF, using *Probabilistic Robotics* as reference material [12]; however, student work is centered on the UKF. By implementing the UKF themselves, students taking this course will be prepared to understand and work with state estimation systems either in academia or in industry.

The EKF is widely used for Micro Aerial Vehicles to estimate state, including acceleration, velocity, and position [13]. It combines data from multiple sensors to form state estimates, fitting the use case for the PiDrone. The UKF achieves better estimation performance while remaining no more computationally intensive than the EKF [14].

The PiDrone software stack implements the UKF due to its performance benefits. Additionally, unlike the EKF, the UKF does not require computing derivatives, making the algorithm more accessible to introductory robotics students.

As the mathematical details of the UKF are quite substantial for an introductory robotics course, we do not require students to implement many of the computations and instead make use of the Python library FilterPy, which has an accompanying online textbook that is presented to students if they desire deeper understanding of the UKF [15]. The project in which students implement the UKF focuses more on the higher-level design decisions and specifications that are necessary to adapt the general UKF algorithm to a particular robotic system such as the PiDrone. Some of these UKF design specifications for our drone, such as which state variables to track and the state transition and measurement functions, are presented.

The Unscented Kalman Filter implemented on the drone calculates prior state estimates, computed in the *prediction* step, and posterior state estimates, computed in the *measurement update* step. Due to the computational complexity of the UKF, two variants were developed to run on the drone: first, a UKF consisting of a simple one-dimensional model of the drone's motion to estimate its position and velocity along the vertical axis, and second, a model that encompasses three spatial dimensions of motion. To differentiate between these two UKFs, we refer to them by the dimension of their state vectors: the simpler model tracks a two-dimensional (2D) state vector, while the more complex model estimates a seven-dimensional (7D) state vector.

1) Two-Dimensional UKF: The 2D UKF has the state vector $\mathbf{x}_t$ shown in Equation (1), which tracks position and velocity along the z-axis.

$$\mathbf{x}_t = \begin{bmatrix} z & \dot{z} \end{bmatrix}^\mathsf{T} \tag{1}$$

To carry out the prediction step of the UKF algorithm, we use a control input $\mathbf{u}_t = \begin{bmatrix} \ddot{z} \end{bmatrix}$, which is the linear acceleration along the z-axis measured by the Inertial Measurement Unit (IMU) onboard the drone. It has been shown that in certain cases the incorporation of IMU data in the measurement update step may be more accurate [16]; however, for relative ease of implementation, we have chosen to treat accelerations as control inputs. This choice is an example of a design decision that students are made aware of when implementing the UKF.

The state transition function, with inputs of the previous state estimate $\mathbf{x}_{t-\Delta t}$, the control input $\mathbf{u}_t$, and the time step Δt, is shown in Equation (2) following one-dimensional kinematics [17].

$$g(\mathbf{x}_{t-\Delta t}, \mathbf{u}_t, \Delta t) = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \mathbf{x}_{t-\Delta t} + \begin{bmatrix} \frac{1}{2}(\Delta t)^2 \\ \Delta t \end{bmatrix} \mathbf{u}_t \tag{2}$$

After a prior state estimate is calculated with the state transition function, the algorithm moves to the measurement function to incorporate sensor measurements. For the 2D UKF, we only consider the drone's downward-facing infrared range sensor—which provides a range reading r—in the measurement update step, so our measurement vector is

given by $\mathbf{z}_t = \begin{bmatrix} r \end{bmatrix}$. The measurement function, then, transforms the state vector into measurement space by selecting the z position component, as shown in Equation (3) [17] .

$$h(\bar{\mathbf{x}}_t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \bar{\mathbf{x}}_t \tag{3}$$

To gain an understanding of the role of the covariance matrices involved in the UKF algorithm, students collect data to characterize the sample variance σ_r^2 of their infrared range sensor. The software stack includes simple data simulation to aid students and researchers as they tune parameters such as the covariance matrices.

2) Seven-Dimensional UKF: The 7D UKF tracks motion in three spatial dimensions with the state vector in Equation (4), as well as the drone's yaw angle ψ.

$$\mathbf{x}_t = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \psi \end{bmatrix}^\mathsf{T} \tag{4}$$

We define a control input $\mathbf{u}_t = \begin{bmatrix} \ddot{x}^b & \ddot{y}^b & \ddot{z}^b \end{bmatrix}^\mathsf{T}$ populated by linear accelerations from the IMU in the drone's body frame. These accelerations are transformed into the global coordinate frame by taking into account the drone's estimated yaw from the state vector, as well as its roll and pitch angles, which are filtered by the IMU. This transformation is carried out with quaternion-vector multiplication as shown in Equation (5), where $\mathbf{q}$ is the quaternion that rotates a vector from the body to the global frame. The use of quaternions—which appear frequently in robotics—in the state transition function offers students an introduction to this mathematical representation.

$$\mathbf{u}_t^g = \mathbf{q} \cdot \mathbf{u}_t \cdot \mathbf{q}^* \tag{5}$$

The resulting global-frame accelerations $\mathbf{u}_{\ddot{x}^g}, \mathbf{u}_{\ddot{y}^g}, \mathbf{u}_{\ddot{z}^g}$ of $\mathbf{u}_t^g$ are used in the state transition function in Equation (6) [17].

$$g(\mathbf{x}_{t-\Delta t}, \mathbf{u}_t, \Delta t) = \mathbf{x}_{t-\Delta t} + \begin{bmatrix} \dot{x}\Delta t + \frac{1}{2}\mathbf{u}_{t,\ddot{x}^g}(\Delta t)^2 \\ \dot{y}\Delta t + \frac{1}{2}\mathbf{u}_{t,\ddot{y}^g}(\Delta t)^2 \\ \dot{z}\Delta t + \frac{1}{2}\mathbf{u}_{t,\ddot{z}^g}(\Delta t)^2 \\ \mathbf{u}_{t,\ddot{x}^g}\Delta t \\ \mathbf{u}_{t,\ddot{y}^g}\Delta t \\ \mathbf{u}_{t,\ddot{z}^g}\Delta t \\ 0 \end{bmatrix}, \tag{6}$$

where $\dot{x}$, $\dot{y}$, $\dot{z}$ are components of $\mathbf{x}_{t-\Delta t}$.

The measurement update step uses the measurement vector $\mathbf{z}_t = \begin{bmatrix} r & x & y & \dot{x} & \dot{y} & \psi_{\text{camera}} \end{bmatrix}^\mathsf{T}$, where r is the infrared slant range reading, x and y are planar position estimates from the downward-facing camera, ψ_{camera} is the camera's yaw estimate, and $\dot{x}$ and $\dot{y}$ are velocity estimates from the camera's optical flow. Equation (7) shows the measurement function, which uses roll ϕ and pitch θ angles to transform altitude into slant range.

$$h(\bar{\mathbf{x}}_t) = \begin{bmatrix} \frac{\bar{\mathbf{x}}_{t,z}}{\cos\theta\cos\phi} & \bar{\mathbf{x}}_{t,x} & \bar{\mathbf{x}}_{t,y} & \bar{\mathbf{x}}_{t,\dot{x}} & \bar{\mathbf{x}}_{t,\dot{y}} & \bar{\mathbf{x}}_{t,\psi} \end{bmatrix}^\mathsf{T} \tag{7}$$

If the planar position estimates originate from localization or SLAM, which already act as filters on raw camera data, then it may not be notably beneficial to apply a UKF on top of these estimates; however, the capability for researchers to incorporate such measurements exists in the 7D UKF implementation, and the higher-dimensional state space offers educational insights to students. For the above reason as well as the computational overhead brought on by estimating seven state variables, at present, the 2D UKF is preferred when attempting stable flight. We analyze the performance of the 2D UKF in the Robot Performance section.

B. Localization

The UKF provides state estimation via sensor fusion, but most mobile robots—the PiDrone included—do not include a sensor to directly measure position [12] [2]. As such, the PiDrone software stack implements both localization and SLAM using the particle filter and FastSLAM algorithms described in *Probabilistic Robotics* [12] and the FastSLAM paper [18].

The implementation of Monte Carlo Localization is based on the particle filter according to Thrun et al. [12] with a customized `sample_motion_model`, `measurement_model`, and particle update process to meet our specific hardware setup. We took the idea of `keyframe` from Leutenegger et al. [19] to only perform `measurement_model` updates when necessary: namely, when the drone has moved a significant distance. We use ORB feature detection from OpenCV [20] to provide the sensor data. We implement a simple version of the localization algorithm that is easy to understand for students and fast enough to run onboard the Raspberry Pi.

Our implementation of `sample_motion_model` is a simplified odometry motion model as shown in Equations (8) and (9). The $\delta_x, \delta_y, \delta_\theta$ are the translations and rotation, and the $\varepsilon_{\sigma_x^2}, \varepsilon_{\sigma_y^2}, \varepsilon_{\sigma_{yaw}^2}$ are the noise we added which are Gaussian zero-mean error variables with variances $\sigma_x^2, \sigma_y^2, \sigma_{yaw}^2$. The algorithm keeps track of particle motion, adding noisy translations and rotations to each particle in every frame. We tried using a velocity-based motion model where the velocity is provided by optical flow from the camera [2]; however, the low-cost camera is influenced by many factors such as light and its reflection. Instead, we use the transform between two frames as parameters for the motion model. With feature detection and matching, we can compute the translation between two frames. Also, we assume the height is known which is provided by the infrared sensor. Therefore, we implemented a 2D localization algorithm.

$$\begin{bmatrix} \hat{\delta}_x \\ \hat{\delta}_y \\ \hat{\delta}_\theta \end{bmatrix} = \begin{bmatrix} \delta_x \\ \delta_y \\ \delta_\theta \end{bmatrix} + \begin{bmatrix} \varepsilon_{\sigma_x^2} \\ \varepsilon_{\sigma_y^2} \\ \varepsilon_{\sigma_\theta^2} \end{bmatrix} \tag{8}$$

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \hat{\delta}_x \cdot \cos(\theta) - \hat{\delta}_y \cdot \sin(\theta) \\ \hat{\delta}_x \cdot \sin(\theta) + \hat{\delta}_y \cdot \cos(\theta) \\ \hat{\delta}_\theta \end{bmatrix} \tag{9}$$

The implementation of `measurement_model` is feature-based with known correspondence as described in Chapter 6 of *Probabilistic Robotics* [12]. The algorithm computes the global position (x', y', θ') of particles based on $features$

from the current frame as shown in Equation (10). The $features$ variable is the collection of features that are extracted from the current frame. To narrow down the feature matching space, the `compute_location` function takes features from the current frame to compare with features that are close to the position of the particle.

In the `sample_motion_model`, we add observation noise to the estimated position before we compute the likelihood of estimated position and the current position of the particle. However, the σ^2 in Equation (11) is the variance for measurement which is different than Equation (8) which is for motion. These are empirically tuned values.

Also, we compute the likelihood q of the estimated position of each particle in Equation (12), simply trusting that most groups of features are unique (the drone flies over non-repeating textured surfaces [2]), instead of using the likelihood of landmarks as described in [12]. This process simplified the measurement model so that it can provide position estimation onboard while flying.

$$x', y', \theta' = \texttt{compute\_location}(features) \tag{10}$$

$$\begin{bmatrix} \hat{x}' \\ \hat{y}' \\ \hat{\theta}' \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} + \begin{bmatrix} \varepsilon_{\sigma_x^2} \\ \varepsilon_{\sigma_y^2} \\ \varepsilon_{\sigma_\theta^2} \end{bmatrix} \tag{11}$$

$$\begin{aligned} q = &\texttt{prob}(\hat{x'} - x, \sigma_x) \\ &\cdot \texttt{prob}(\hat{y'} - y, \sigma_y) \\ &\cdot \texttt{prob}(\hat{\theta'} - \theta, \sigma_\theta) \end{aligned} \tag{12}$$

The update process of the Monte Carlo Localization is similar to that from Thrun et al. [12], except that we use a keyframe scheme: the `measurement_model` is called only if the `sample_motion_model` has been processed a certain number of times or the estimated position of the drone is far from the estimate at the last time we performed the `measurement_model` process. This allows the localization algorithm to run in a single thread onboard, which would be realizable for students with less multi-threading background.

C. Simultaneous Localization and Mapping

FastSLAM, described in Montemerlo et al. [18], is a particle filter algorithm for Simultaneous Localization and Mapping. The dominant approach to this problem has been to use an EKF to estimate landmark positions. The chief problem with this approach is computational complexity: the covariance matrix of an EKF-SLAM with N landmarks has at least N^2 entries. Instead, FastSLAM factors the posterior distribution by landmarks, representing each landmark pose with a single EKF. There are some more modern SLAM algorithms such as ORB-SLAM2 [21], which may provide more accurate state estimation in 3D space; however, such algorithms are more complicated to understand for students. FastSLAM is simple to understand and a natural fit for the PiDrone platform as it directly builds off of Monte Carlo Localization. Particles in MC Localization are augmented by adding a landmark estimator (EKF) for each observed landmark to each particle. The motion model, keyframe scheme, and resampling methods are left unchanged from localization. The measurement model is replaced with a `map_update` method to associate observed features with existing landmarks and assign a probability to particles, which each represent an estimate of the robot pose and map.

SLAM seeks to estimate the posterior distribution $p(\Theta, x^t | z^t, u^t)$ where Θ is the map consisting of N landmark poses $\Theta = \theta_1, ...\theta_N$, x^t is the path of the robot $x^t = x_1, ..., x_t$, z^t is the sequence of measurements $z^t = z_1, ..., z_t$, and u^t is the sequence of controls, $u^t = u_1, ..., u_t$. The main mathematical insight of FastSLAM is the ability to factor this distribution by landmark as Equation (13).

$$p(\Theta, x^t | z^t, u^t) = p(x^t | z^t, u^t) \Pi_n p(\theta_n | x^t, z^t, u^t) \tag{13}$$

This approach is sound since individual landmark estimations are conditionally independent, assuming knowledge of the robot's path and correspondences between observed features and landmarks in the map.

The factored posterior is realized with a 2D EKF to estimate each landmark pose in each particle of the filter. Newly observed ORB features [20] are entered as landmarks into the map, and the EKFs of existing landmarks are updated when re-observed. The main difference of FastSLAM from MC Localization, then, is the `map_update` step, described in Algorithm 1, which associates newly observed features with existing landmarks and assigns a weight to each particle. Note that the $landmarks$ list consists of only landmarks within a small radius r of the particle's pose, computed by line 2. The method $get\_perceptual\_range()$ uses the drone's height to compute the radius of the largest circle within full view of the drone's camera. This ensures that the map represented by each particle is conditioned on the unique robot path represented by that particle. Also note that the landmark counter scheme ensures existing landmarks which are not matched to but lie within the camera's field of view are removed in line 27. We determine match quality using Lowe's Ratio Test from [22] in line 10. Line 9 finds the two best-matching landmarks for an observed feature using OpenCV's `knnMatch` [20]. Finally, the weight of each particle is incremented for each landmark within its perceptual range. The weight is increased by an importance factor proportional to the quality of the match for re-observed landmarks in line 18, and the weight is decreased for new landmarks in line 13. The $threshold$ is defined as some constant between 0 and 1.

Given the high density of features required for the motion model, PiDrone SLAM creates landmark-dense maps, resulting in slow map updates. It was found that multithreaded map updates are required to perform SLAM in real time (online SLAM) or else the robot path will get lost if the drone moves during map updates. Performing thread-safe updates to SLAM particles is challenging for introductory robotics students. Rather, we present a method for performing SLAM sequentially (offline SLAM), allowing students to implement the simple FastSLAM

Algorithm 1 SLAM - Map Update

procedure MAP_UPDATE($observed\_features$)
  $r \leftarrow get\_perceptual\_range()$
  for $p \in particle\_set$ **do**
    $landmarks \leftarrow$ empty list
    for $lm \in p.landmarks$ **do**
      if $dist(lm, p.pose) \leq r$ **then**
        $landmarks \leftarrow landmarks \cup lm$
    for $f \in observed\_features$ **do**
      $match1, match2 \leftarrow best\_2\_matches(f, landmarks)$
      if $match1.dist > 0.7 \cdot match2.dist$ **then**
        $lm \leftarrow initEKF(p.pose, f.pose)$
        $lm.counter \leftarrow 0$
        $p.weight \leftarrow p.weight + \log(threshold)$
      else
        $old\_lm \leftarrow landmarks[match1\_idx]$
        $lm \leftarrow updateEKF(p.pose, f.pose, old\_lm)$
        $importance \propto dist1 - dist2$
        $p.weight \leftarrow p.weight + importance$
      $p.landmarks \leftarrow p.landmarks \cup lm$
      $lm.matched \leftarrow true$
    for $lm \in p.landmarks$ **do**
      if $lm.matched$ **then**
        $lm.counter \leftarrow lm.counter + 1$
      else
        $lm.counter \leftarrow lm.counter - 1$
      if $lm.counter < 0$ **then**
        $p.landmarks \leftarrow p.landmarks \setminus lm$

algorithm. We collect flight data on a flying drone, then perform SLAM offline on the saved data to build a map. The drone then flies, performing MC localization over the map created from SLAM; this runs in real time onboard the drone. The method shows that maps converge correctly by localizing over a map created by FastSLAM.

IV. ROBOT PERFORMANCE

The aim of our validation was to assess the performance of the system relative to ground truth, in order to show that we are accurately able to perform state estimation. Fig. 3 depicts the testing environment which includes: the flying space enclosed in safety netting, the motion capture cameras, a highly textured planar surface to fly over, the drone with reflective markers, and the computer used for offboard computations.

A. Unscented Kalman Filter

The 2D UKF was run onboard the Raspberry Pi with a predict-update loop executing at 30 Hz to provide the drone with estimates of its altitude. We compared these filtered estimates to the raw infrared range readings and to the smoothed

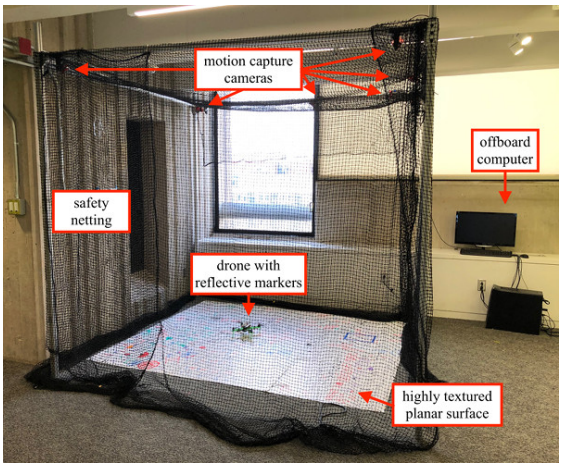


Fig. 3: Environmental setup used for evaluation.

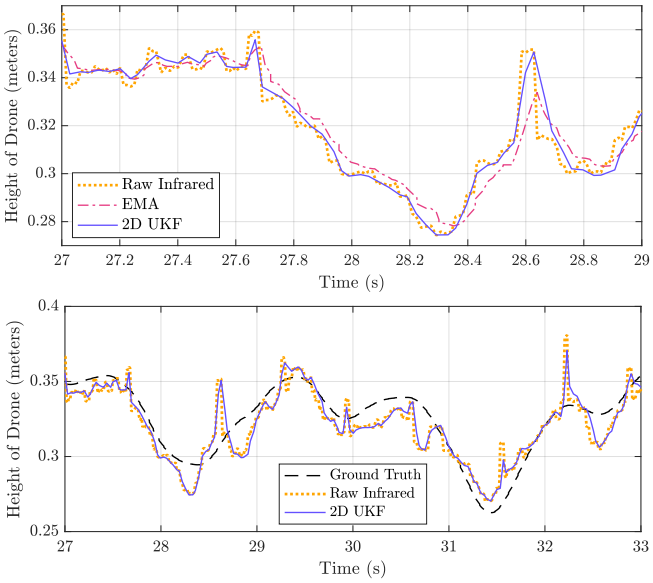


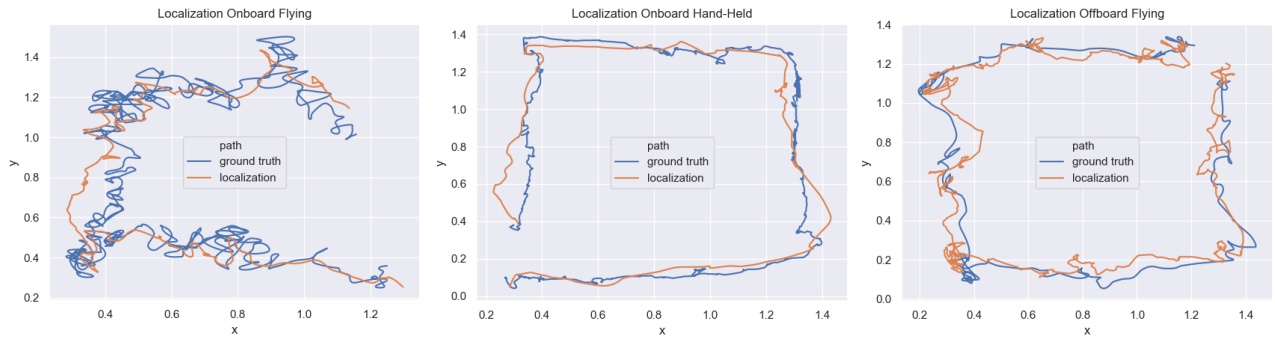
Fig. 4: Height estimates of 2D UKF against other sources.

readings from the Exponential Moving Average (EMA) filter, which was the filtering method used before the development of the UKF. However, the latency inherently introduced by an EMA filter was considered undesirable for a quadrotor. Fig. 4 displays the results of a flight test in which the drone was commanded to hover in place. The 2D UKF curve follows the raw infrared readings with less latency than the EMA in quick ascents and descents. Additionally, the UKF estimates demonstrate smaller fluctuations than the noisy infrared sensor. Although the UKF estimates require more computation than a naive EMA, its benefits are apparent in Fig. 4 and could be particularly useful for agile maneuvers.

The 2D UKF was also tested against ground-truth height provided by a motion capture system. The results of this comparison are shown in Fig. 4.

B. Localization

We set up the motion capture system to collect the x and y coordinates of the drone at 120 Hz. The onboard localization



(a) (b) (c)

Fig. 5: Localization algorithm runs in three conditions

	Mean	Std	Maximum	Minimum
Onboard Flying	0.115	0.062	0.301	0.007
Onboard Hand-Held	0.096	0.049	0.261	0.010
Offboard Flying	0.074	0.042	0.190	0.007

TABLE I: Error between our Localization implementation and ground truth from motion capture system (meters).

is running at 5 Hz with 40 particles, collecting 180 features at each frame along with 19,200 map features. The offboard localization is running at 12 Hz with 50 particles, collecting 250 features at each frame along with 48,000 map features. We use data from the motion capture system as the ground truth to validate our localization implementation. The drone was flying over a 1.67m $\times$ 1.65m textured surface, and we collected data while the drone was flying in a square. To prepare the map for localization, we took photos of the surface around 55 centimeters, and stitched them using Image Composite Editor from Microsoft [23]. There are some distortions of the map image which causes offsets for a certain area, but our localization still works despite these distortions.

The localization running onboard while the drone is flying is compared to ground truth in Fig. 5a. When running localization onboard the drone and flying, the Raspberry Pi spends less of its computational resources on the PID controller, resulting in less stable flight. This is due primarily to the computational load of the `measurement_model`. To accurately evaluate the quality of the localization algorithm, we also collected data while moving the drone by hand. The results are shown in Fig. 5b. We moved the drone slowly and steadily, and the localization algorithm estimated the position close to the ground truth. Those two images show that our simplified localization implementation is functional. Also, if students want to do further research with the PiDrone, the offboard version of localization provides higher accuracy and more stable flight as shown in Fig. 5c, allowing other research code to run simultaneously.

Table I shows statistics for data we collected. We compared coordinates from our localization implementation to the ground truth and we pair them based on the Robot Operating System (ROS) timestamp. The error for one pair would be calculated as

$$\text{error} = |x - x'| + |y - y'|,$$

where x, y are the planar coordinates from the localization algorithm and x', y' are the planar coordinates from the motion capture system. Considering that the drone is flying and we did not account for the angle while the frame is captured, the mean error is acceptable. With more computing resources and stabler flying, the accuracy is higher.

	Mean	Std	Maximum	Minimum
Offboard Hand-Held	0.127	0.0677	0.265	0.012

TABLE II: Error between our SLAM implementation and ground truth from motion capture system (meters).

C. Simultaneous Localization and Mapping

Data are collected from offline SLAM (see Section III-C for motivation) and compared to ground truth. Initially, the drone was moved by hand over a 1.67m x 1.65m highly textured planar surface, extracting 200 ORB features per frame at a rate of 30 Hz and saving them to a text file, as well as saving an infrared height reading with every image frame. Then, SLAM was performed with 40 particles onboard the grounded drone using the saved flight data, resulting in a map containing 5,108 landmarks. Finally, the drone was moved by hand over the same textured surface, performing MC localization on an offboard machine with 20 particles using the map created by SLAM. These poses are obtained at a rate of 14 Hz. We compare the pose data from offline SLAM with ground-truth obtained by a motion capture system. Table II gives the mean, standard deviation, maximum, and minimum error. Fig. 6 plots the pose estimate from offline SLAM compared to ground truth.

V. IMPACT IN EDUCATION

The first run of the PiDrone course by Brand et al. [2] demonstrated the potential of the platform and course. Simplified versions of the course have been used to introduce robotics to dozens of high school students. At the Providence Career and Technical Academy, the PiDrone has been used as part of the engineering curriculum for two years. The

Fig. 6: Hand-Held Offboard Localization over SLAM-created map.

course has been taught at a Summer@Brown session to high school students from across the nation and world putting autonomous flight in the hands of students of diverse backgrounds. The course was also offered at a rural public school in Upstate New York, again proving the ability of the course to empower students of all backgrounds with their own autonomous robots. The PiDrone course for high school students will be taught again at Summer@Brown and at MassRobotics, a local robotics incubator in Boston, Massachusetts [24].

Increasing platform accessibility, the PiDrone platform has merged with Duckietown under the name of Duckiesky [25]. Duckietown is a popular platform for learning autonomy on ground robots [25]. This merge resulted in the creation of an online textbook which makes the platform, learning materials, and operation instructions readily accessible. These new online resources were used for the Fall 2018 course at Brown University. The course proved successful with 23/25 students covering the rigorous content and implementing their own algorithms for autonomy. The greatest challenge of the course is introducing a broad array of robotics concepts within a time frame that only allows students to scratch the surface. Despite this challenge, upon completion of the course, students are well-equipped with the tools needed to approach problems in autonomous robotics.

The future outlook of the platform as an educational tool includes expansion into additional universities as an undergraduate course, and the creation of online learning modules and additional projects for learning and building upon the drone's current capabilities.

VI. FUTURE WORK

The PiDrone is still under heavy development. We are still working on the drone's stability and safety, as well as increasing functionality.

We plan to continue expanding the PiDrone platform, particularly its autonomous capabilities. Huang et al. have performed research [26] using the PiDrone with Mixed Reality, natural language commands, and high-level planning. Adding high-level planning and support for Markov Decision Processes allows students to learn and work with reinforcement learning, which can build upon the work of Huang et al. For hardware updates, we would like to replace the current IR sensor with a sensor that can more accurately estimate the drone's height at higher altitudes. Further exploration of adding a forward-facing camera to the drone will enable the implementation of SLAM in three dimensions.

The addition of a UKF, localization, and SLAM to the drone demonstrates that the drone can support higher level autonomy, and opens the door for implementing more advanced autonomous functionality. Continued work regarding the drone's vision capabilities will permit object tracking and motion planning tasks. Generating a proper dynamics model of the drone will allow for the implementation of advanced control algorithms that are better suited than a PID controller for performing aggressive maneuvers such as acrobatic flipping or perching. Based on the expanding capabilities of the platform, new instructional projects will be created for future iterations of the educational course, which will run again in Fall 2019 at Brown University.

The course will also be integrated into the edX platform [27] so that it can be taught both online and in residential settings. A crowdfunding campaign is planned to enable packaging of the drone parts into self-contained kits to distribute to individuals who desire to learn autonomous robotics using the PiDrone platform. With the modularized software architecture and existing capabilities of the drone, students, educators, and researchers alike can easily use and build upon the PiDrone platform to explore autonomy in aerial robotics.

VII. CONCLUSION

Current educational robots do not exhibit significant autonomous abilities. Advancing the PiDrone, we present a low-cost educational drone platform for an introductory robotics course that teaches advanced algorithms for state estimation in an accessible way. Although implemented on a Raspberry Pi in Python, the performance of the UKF, MC localization, and FastSLAM on the PiDrone makes the platform a compelling framework for introducing students of any background to robotics and high-level autonomy.

ACKNOWLEDGMENT

We would like to thank Amazon Robotics for their donations which help fund development and equipment for the course and platform.

This work is supported by the National Aeronautics and Space Administration under grant number NNX16AR61G.

We would like to thank Duckietown for their technical support.

We would also like to thank James Baccala, Jose Toribio, and Caesar Arita for their contributions to the drone hardware platform. Special thanks go to James for designing the propeller guards.

References

[1] International Data Corporation. Worldwide spending on robotics and drones forecast to accelerate over the next five years, reaching $201.3 billion in 2022, according to new idc spending guide, jul 2018. URL https://www.idc.com/getdoc.jsp?containerId=prUS44150218. Accessed: March 1, 2019.

[2] Isaiah Brand, Josh Roy, Aaron Ray, John Oberlin, and Stefanie Tellex. Pidrone: An autonomous educational drone using raspberry pi and python. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–7. IEEE, 2018.

[3] dji.com. Tello edu. URL https://store.dji.com/product/tello-edu?vid=47091. Accessed: February 17, 2019.

[4] skydio.com. Skydio. URL https://www.skydio.com/. Accessed: February 17, 2019.

[5] Introduction to robotics course. URL http://cs.brown.edu/courses/cs1951r/.

[6] Luke Eller, Théo Guérin, Baichuan Huang, Josh Roy, Stefanie Tellex, Garrett Warren, and Sophie Yang. Duckiesky learning materials, . URL https://docs-brown.duckietown.org/doc-sky/out/introduction.html. Access credentials: username=brown, password=imdown.

[7] Luke Eller, Théo Guérin, Baichuan Huang, Josh Roy, Stefanie Tellex, Garrett Warren, and Sophie Yang. Duckiedrone operations manual, . URL https://docs-brown.duckietown.org/downloads/opmanual_sky/docs-opmanual_sky/branch/master/opmanual_sky/out/build.html. Access credentials: username=brown, password=imdown.

[8] Introducing raspberry pi hats, Feb 2015. URL https://www.raspberrypi.org/blog/introducing-raspberry-pi-hats/.

[9] Open-source flight controller software for modern flight boards. URL http://cleanflight.com/.

[10] Crazyflie 2.1. URL https://www.bitcraze.io/crazyflie-2-1/.

[11] Betaflight. Are you ready to fly? URL https://betaflight.com/.

[12] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623.

[13] Udacity. Flying car nanodegree program. URL www.udacity.com/Flying-Car. Accessed: February 14, 2018.

[14] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, pages 153–158. Ieee, 2000.

[15] Roger R Labbe Jr. Kalman and bayesian filters in python, May 2018. URL https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python.

[16] A. T. Erdem and A. . Ercan. Fusing inertial sensor data in an extended kalman filter for 3d camera tracking. *IEEE Transactions on Image Processing*, 24(2):538–548, Feb 2015. ISSN 1057-7149. doi: 10.1109/TIP.2014.2380176.

[17] S. Tellex, A. Brown, and S. Lupashin. Estimation for Quadrotors. *ArXiv e-prints*, August 2018.

[18] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 2002.

[19] Stefan Leutenegger, Paul Furgale, Vincent Rabaud, Margarita Chli, Kurt Konolige, and Roland Siegwart. Keyframe-based visual-inertial slam using nonlinear optimization. *Proceedings of Robotis Science and Systems (RSS) 2013*, 2013.

[20] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[21] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017. doi: 10.1109/TRO.2017.2705103.

[22] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[23] Image composite editor. URL https://www.microsoft.com/en-us/research/product/computational-photography-applications/image-composite-editor/.

[24] We bring robotics to life. URL https://www.massrobotics.org/.

[25] Duckietown a playful way to learn robotics. URL https://www.duckietown.org/.

[26] Baichuan Huang, Deniz Bayazit, Daniel Ullman, Gopalan Nakul, and Stefanie Tellex. Flight, Camera, Action! Using Natural Language and Mixed Reality to Control a Drone. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[27] edx. URL https://www.edx.org/.