The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

Autonomous Indoor Helicopter Flight using a Single Onboard Camera

Sai Prashanth Soundararaj and Arvind K. Sujeeth
Electrical Engineering Department
Stanford University, USA
{saip,asujeeth}@stanford.edu

Ashutosh Saxena
Computer Science Department
Cornell University, USA
asaxena@cs.cornell.edu

***Abstract*— We consider the problem of autonomously flying a helicopter in indoor environments. Navigation in indoor settings poses two major challenges. First, real-time perception and response is crucial because of the high presence of obstacles. Second, the limited free space in such a setting places severe restrictions on the size of the aerial vehicle, resulting in a frugal payload budget.**

We autonomously fly a miniature RC helicopter in small known environments using an on-board light-weight camera as the only sensor. We use an algorithm that combines data-driven image classification with optical flow techniques on the images captured by the camera to achieve real-time 3D localization and navigation. We perform successful autonomous test flights along trajectories in two different indoor settings. Our results demonstrate that our method is capable of autonomous flight even in narrow indoor spaces with sharp corners.

I. Introduction

Unmanned aerial vehicles have several indispensable applications in surveillance, intelligence and search and rescue (SAR) operations. We address the problem of autonomous flights in indoor environments, which often have only limited free space and may contain a number of obstacles. These constraints limit the potential size of the aerial vehicle.

Miniature radio controlled helicopters are particularly suited for indoor navigation due to their low-speed flight and in-place hovering capabilities. However, their small size places a frugal payload restriction on weight and power supply. Miniature cameras are an attractive option because they are small, passive sensors with low power requirements. In comparison, other sensors such as lasers (an active sensing method, e.g., Hokuyo [11]) are heavy (450g) and have high power requirements (500mA). In our work, therefore, we use a miniature light-weight camera (Fig. 1) as the sensor.

Related work on autonomous helicopter flight in outdoor settings such as [1], [8], [9], [5] have led to terrific control algorithms capable of performing extreme aerobatic maneuvers. However, navigation in indoor environments pose complementary challenges both in the perception and control aspects:

- Indoor navigation is more of an accurate perception and navigation problem than a control problem; it is crucial to be able to navigate within close quarters of obstacles in real-time.
- The smaller RC helicopters are limited in their payload capacities (less than 70g). This allows only light-weight sensors (such as small cameras) to be used.

Fig. 1. Our miniature indoor helicopter (13.6in rotor diameter, 227g weight). Inset on the right shows the miniature wireless camera used.

- Images from the on-board video capture are subject to severe vibrations due to helicopter motion.

To address these issues, we develop a purely vision-based system, where we fly the helicopter autonomously using only images from a single on-board camera. We consider the following scenario: there is a ground robot that moves around the environment, takes images and builds a map of the environment (e.g., [22]). For example, we earlier used our STanford AI Robot (STAIR) to navigate and take pictures in our building [13]. Now, our miniature helicopter can take the images from an on-board camera and fly autonomously in these environments using this database in real-time. We present an algorithm which uses a non-parametric learning algorithm (K-nearest neighbor with cover trees implementation, see Section IV-B for perception). The output of this algorithm is used in a PD (proportional differential) controller to control the helicopter.

Our algorithm and controller framework are capable of autonomously maintaining stable in-place hover of the helicopter, as well as following user-defined trajectories in environments with limited free space. It is also capable of correcting for drifts in motion, enabling it to maneuver in narrow regions without crashing into boundaries.

We test our algorithm in two different environments. One of them has sharp corners and obstacles along the sides of a confined space and the robot has to correct for drift to remain in the free space. The other has free space but few

discriminative features. We show that using our image-based algorithm, our helicopter is able to fly successfully within user-defined trajectories in both these environments.

II. Related Work

There is a long and distinguished list of prior work on autonomous helicopters in outdoor environments. We refer the reader to a few papers ([1], [8], [9], [5]) because our problem of indoor flight addresses different issues.

Several researchers have previously performed indoor flight experiments. However, these are usually conducted in uncluttered open spaces and using the more stable quad-rotor helicopter models. For example, Tournier et al. [25] used known patterns (Moire patterns) pasted in the environment to estimate the attitude and position of quad-rotor vehicles. He et al. [10] flew a quad-rotor in GPS denied indoor environments, using a Hokuyo laser (a device that gives 3D depth). Roberts et al. [20] used infra-red and ultrasonic sensors to fly a quad-rotor indoor in a large (7×6m) room, while [12] used vision to make a quad-rotor hover stably in an indoor environment. Quad-rotors, however, have larger dimensions (e.g., 730mm in span for [3], 550mm for [20]) that make it harder for them to navigate in constrained spaces.

Mejias et al. [14] used vision to land a helicopter while avoiding power lines. Nicoud and Zufferey [17] discussed the tradeoffs of designing indoor helicopters while D. Schafroth et al. [21] designed various test benches for micro helicopters and designed a dual-rotor single-axis helicopter with an omnidirectional camera. Mori et al. [16] used markers to stably hover a co-axial helicopter and go from one marker to another.

In other related work, Michels et al. [15] used an on-board camera for autonomous obstacle avoidance in a small RC car driving at high speeds. They compute image features by convolving the image with a number of filters, and used linear regression to predict distances to obstacles.

One can also build a map of the environment using Visual SLAM (e.g., [27], [6], [22]), in which the map is built using images captured from a camera. Ribnick et al. [19] estimate positions and velocities from monocular views. Such methods for estimating position/location from visual landmarks have been used in many robots which navigate on 2D ground.

In comparison to ground vehicles, aerial vehicles have additional challenges posed by their 3D movement and the additional degrees of freedom in helicopter orientation. Furthermore, in the case of aerial navigation in highly constrained spaces, we require very fast response times; additional computation (e.g. feature computation) can prohibitively increase the latency of the controller.

Our method of using images captured from a miniature camera could be applied to autonomous flights of other small aerial robots as well. For example, extremely small robots such as 25mm long micro-mechanical flying insects [7] can use miniature cameras.

Fig. 2. The controller setup showing the wireless image receiver, the laptop and the wireless transmitter.

III. Helicopter Platform

Our test platform is based on the E-flite Blade CX2 and Blade CX3 coaxial helicopters (Fig. 1). These are cheap, ready-to-fly micro helicopters. Each helicopter's rotor diameter is 0.36m (with the landing gear in place) – this enables it to fly even in confined spaces. However, its total weight is 227g and allows a payload with a maximum weight of only 70g. We chose the coaxial model because it provides stable flight and is considerably easier to control than its dual rotor counterparts. A Swan EagleEye wireless camera is mounted on-board (Fig. 1 inset) to stream images in real-time to the control algorithm. The camera is small ($22 \times 24 \times 27$ mm), light-weight (14g), and has low power consumption (100 mA). It relays back images of 640 x 480 resolution at 30 frames per second. This camera is our *only* perception source for the helicopter control – we do not even use inertial sensors in our algorithm.

A wireless receiver attached to the laptop is used to receive the streaming frames. We use a commercial interface, Endurance's PCTx, to link the transmitter to the USB port of the laptop, via the trainer port (see Fig. 2). This enables the helicopter to be controlled via the PC using our control algorithm. We use the Spektrum DX6i 2.4 GHz transmitter, which is capable of translating PPM input to DSM signals that are broadcast to the helicopter.

IV. Visual Perception

There are various methods one can use for visual perception, such as using SIFT features or dense 3D reconstruction. While SIFT features can provide additional robustness, it comes at the expense of increased computation time. In our real-time application, any elaborate computation is unsustainable. Furthermore, since SIFT features tend to be specific, they have little hope of generalizing to similar but unknown environments. With these limitations in mind, we chose a simpler strategy that can still provide efficient and robust perception.

For image-based localization, we use a non-parametric learning algorithm (nearest-neighbors implemented with cover trees, described below). It gives us a "global" prediction of the state Ψ (location and orientation) of the helicopter. We will also use a scoring function for taking into account helicopter dynamics while predicting the location. Finally, we obtain velocity estimates using optical flowi. Our PD controller combines the two predictions (global location estimate and velocity estimate) to generate commands for the helicopter.

For the purposes of flying a robot indoors, its (x, y) location, its height z and the angle θ (yaw) to which the helicopter points are sufficient to describe a hovering location. The other two orientations of the helicopter ψ (pitch) and γ (roll) are close to zero in the stable state of the helicopter, and become non-zero only when the helicopter is controlled to move to a new location. Therefore, we describe the state of the helicopter as $\Psi = (x, y, z, \theta)$.

A. Image database using Cover Trees

A cover tree [2] is a computationally-efficient data structure for performing nearest neighbor lookups from a large database. (There are many other implementations for fast nearest neighbor lookups, e.g. KD trees [18]; cover trees are most suited for our purpose because they are not as limited by the number of dimensions.) It represents the data points in a tree structure with storage size linear in the number of data points. This enables one to store visual maps of a large scale environment, such as a building.

Database retrieval can be performed using a number of methods. Cummins and Newmann [6] used interest points on small image patches as the basis of representation. This method is useful for recognizing a specific image feature in the database. Since our images contain extensive motion blur due to helicopter motion, it is harder to recognize individual features. Therefore, we use the whole image as the basis of the representation.

In the field of computer vision, Torralba [24] presented the idea of tiny images for large databases. They started with 80 million images from the Internet, and despite projecting it down to only a few dimensions, they were still able to recognize objects given a new image. Motivated by [24] and [26], we reduce the dimension of a full image from our camera from 307200 pixels to 32 using Principal Components Analysis (PCA) (See Fig. 3). This significantly reduces the storage and computational requirements. Each vector $q \in \mathbb{R}^{32}$ for a frame is stored as a point in the cover tree $\mathcal{C}$. This allows us to perform fast classification of an incoming test frame by searching for the K-Nearest Neighbors using its PCA projection. We chose the K-Nearest Neighbors (kNN) algorithm to classify a frame with respect to its position and attitude because it is a fast, data-driven approach.

B. Prediction with spatial proximity

Visually, two images corresponding to different locations often look very similar even if they are of two spatially distinct areas. For example, one hallway would look exactly the same as another. In order to disambiguate, we maintain a history $\mathcal{H}$ of the past few states (five in our experiments) of the helicopter. Since we know that the helicopter cannot change its location drastically in a short interval, we can use this memory of past states to improve the estimation of the helicopter's location. In detail, when a new frame is received from the helicopter's camera, we extract the K nearest neighbors ($n_1, n_2, \ldots, n_K$, with labels $\Psi_1, \Psi_2, \ldots, \Psi_K$ ordered from closest to farthest) from its PCA projection using cover trees. We compute the score J for each of the $i = 1, 2, ..., K$ nearest neighbors.

$$J(\Psi_i) = \sum_{j \in \mathcal{H}} ||\Psi_i - \mathcal{H}_j||_2 \tag{1}$$

where $\mathcal{H}_j$ represents the j^{th} image in the image history. We then define the Ψ_i with the lowest (best) score J as

$$\Psi^* = \arg\min_i J(\Psi_i) \tag{2}$$

to be the current prediction (with corresponding nearest neighbor n^*).[1]

However, in many cases, our image-based classification is not unimodal in that it needs to maintain multiple beliefs about its state. For example, consider a helicopter flying in a hallway and facing a wall; turning either left or right would result in a similar visual appearance. An algorithm that uses Kalman filters also suffers from this problem when the belief about the state is not unimodal. Particle filters [23] provide a nice method to address this issue; however, they are computationally intensive. As a simplification of these methods, we account for the case where the helicopter has high confidence in its state due to a location with a visually discriminative feature. We therefore estimate the state as

$$\hat{\Psi} = \arg\min(\nu||n^* - q||_2, ||n_1 - q||_2) \tag{4}$$

where q is the query point and ν is a coefficient between 0 and 1 indicating the confidence in the closest point, i.e., a parameter describing the relative confidence of the values

Fig. 3. Image frames before and after Principal Component Analysis.

[1] In practice, we take three neighbors ($i = 1, 2, 3$) with lowest score J, and define the expected state as

$$\Psi' = \sum_i w_i \Psi_i, \tag{3}$$

where $\sum_i w_i = 1$. Using Ψ' instead of Ψ^* mitigates a number of errors caused by noise.

obtained from the spatial proximity estimate n_1 as compared to the nearest neighbor value n^*. This estimate is then taken to be the prediction of the current state Ψ of the helicopter.

C. Optical Flow

Self-created turbulence and random drifts are a common occurrence when flying miniature RC helicopters. To help stabilize the helicopter against these unwanted movements, we use the optical flow across images to estimate the helicopter's current velocity and direction of motion. This velocity estimate is then used by the control algorithm to stabilize the helicopter as described in Section IV-D. The main idea is to provide a counter-acting effect that helps prevent the helicopter from turning or moving too drastically in a short period, giving it additional stability.

Optical flow is implemented using 400 features and OpenCV's implementation of the Pyramidal Lucas Kanade (PLK) algorithm [4]. We calculate the total optical flow as the mean over all the features for each frame (we additionally apply a median filter in time to remove outliers). Fig. 4 shows the calculated optical flow on an example frame, and the inset shows the attitude estimated by kNN. We sum the horizontal components of the optical flow in the horizontal direction to get O_{row}, and sum in the vertical direction to get O_{col}.

Fig. 4. Optical flow computed from a typical image.

D. Control Algorithm

The inputs to our control algorithm are the image frames relayed in real-time from the on-board wireless camera. Our algorithm uses these to estimate the current position and velocity of the helicopter and then outputs those estimates to a PD controller.

We will first describe how we use the vision output to generate desired velocity controls for the helicopter. Let the current state of the helicopter be Ψ; the noisy estimate of the state obtained using our kNN algorithm is $\hat{\Psi} = (\hat{x}, \hat{y}, \hat{z}, \hat{\theta})$; and the desired values (e.g. x_{desired}) are the coordinates that we want to move towards. We combine the output of vision (current location from kNN and velocity estimate from optical flow) to stabilize the helicopter as:

$$\dot{x}^* = \alpha_1 * (x_{\text{desired}} - \hat{x}) + \beta_1 * \hat{\dot{x}}_{\text{OF}} \quad (5)$$

$$\dot{y}^* = \alpha_2 * (y_{\text{desired}} - \hat{y}) + \beta_2 * \hat{\dot{y}}_{\text{OF}} \quad (6)$$

$$\dot{z}^* = \alpha_3 * (z_{\text{desired}} - \hat{z}) + \beta_3 * \hat{\dot{z}}_{\text{OF}} \quad (7)$$

$$\dot{\theta}^* = \alpha_4 * \left(\theta_{\text{desired}} - \hat{\theta}\right) + \beta_4 * \hat{\dot{\theta}}_{\text{OF}} \quad (8)$$

where $\hat{\dot{x}}_{\text{OF}} = O_{\text{row}} \cos\theta$, $\hat{\dot{y}}_{\text{OF}} = O_{\text{row}} \sin\theta$ and $\hat{\dot{z}}_{\text{OF}} = O_{\text{col}}$ are the velocity estimates calculated by optical flow in x, y and z-directions, respectively. α and β are tunable parameters, estimated through controlled experiments.

By varying the desired (or goal) values continuously, we can make the helicopter follow trajectories and maintain orientations in space. For example, when the helicopter is at its desired location, the first term in each of the equations would be close to zero, and the second optical flow term would prevent the helicopter from moving due to a few misclassifications.

We can give four control commands to the helicopter—throttle T (mostly changes z), rudder R (mostly changes θ), elevator E (mostly moves the helicopter forward/backward) and aileron A (mostly moves the helicopter left/right). We write a control command as $C = (A, E, T, R)$. Note that the "stable" control commands are $(0, 0, 0, 0)$, i.e., the state of the helicopter with correct trim settings. A non-zero control would give the helicopter a velocity in a particular direction. We compute the commands to be given to the helicopter from the desired velocity $\dot{\Psi}^*$ as

$$C = M\dot{\Psi}^* \quad (9)$$

where $M \in \mathbb{R}^{4\times4}$ is the control mixing matrix.[2]

E. Data Collection

Our method uses a data-driven supervised learning algorithm for estimating the current state ψ of the helicopter. This necessitates a labeled training data set. Our training data consists of videos taken along uniformly spaced points in each environment. At each point, we take a training video by rotating first clockwise and then counter-clockwise at approximately uniform velocity, which provides a continuous set of θ labels at a particular (x,y) coordinate. We also take training videos at a fixed θ along a line of points in both x and y directions at varying heights z.[3] The data points are then labeled assuming that the motion was uniform during each training video.

To construct our final training datasets, we additionally apply a Gaussian blur to each frame of the training videos and add those to the training set along with the normal

[2] M is mostly a diagonal matrix with the gains hand-tuned. Only a few off-diagonal entries are non-zero, (i.e., the trim correlations), for example, changing elevator mostly moves the helicopter forward/backward but also slightly changes z. Finally, since our representation (x, y) is in global coordinates, the first 2×2 components of M have $[\cos\theta, \sin\theta; \quad -\sin\theta, \cos\theta]$.

[3] We use a combination of different initial orientations to reduce the labeling error due to imprecision in the data collection process.

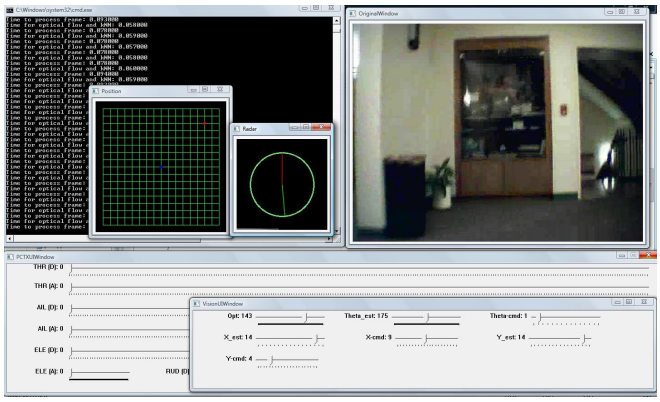


Fig. 5. Controller interface showing location estimate and control signal values.

unblurred frames. This helps account for varying amounts of motion blur (which depends on the velocity of the helicopter) in the in-flight video stream.

We often encountered significant problems with radio frequency interference in the wireless receiver for the on-board camera. To mitigate this, we used a fast and reasonably accurate filter based on the histogram of an image's edges. We applied the noise filter to both the training sets and the incoming frames in real-time.

V. Controller Interface and Simulator

A. Controller Interface

The interface was designed to provide us flexibility in running experiments, having a smooth control of navigation and effective realtime feedback. We use Endurance's low level PCTx driver for communicating through the radio-controller interface. Fig. 5 shows a screenshot of the controller during an actual flight. The display presents several pertinent real-time results to the user: the video stream from the helicopter's wireless camera, and displays showing the current kNN estimates for location and orientation, optical flow value and command signals generated by the control algorithm.

B. Simulator

We implemented a simple simulator of helicopter behavior to test our vision algorithm (Fig. 5). The simulator builds a database to model the spatial structure of the surroundings based on a test video. It interacts with the vision and control framework by accepting a control command, retrieving the image frame that would have resulted if the command was issued to the helicopter in the actual setting, and passing the frame to the controller.[4] The retrieval is done by searching for the image in the test video closest to the estimated state. We also simulate the latency of the helicopter response due to communication/processing delays and physical effects such as inertia. The simulator enables us to both verify classification accuracy along trajectories and visualize the controller's dynamics.

[4]The resulting image frame is an approximation, since it is constrained to be a frame present in the test video.

Fig. 6. An image showing our test environment E1.

VI. Experiments

We tested our algorithm in two different locations. Our first environment (E1) is a relatively feature-less open rectangular space (4.8×4.8m). This environment provides few obstacles for the helicopter, but poses a challenging localization problem. Conversely, our second environment (E2) is in a constrained space (1.8×3.6m) with narrow corridors (1.3m wide) and a sharp turn, where even minor deviations in flight can cause a crash. We collected 28306 and 31559 training frames in E1 and E2 respectively. After projecting to 32 dimensions using the PCA model, storing the training model required only 9.51 MB and 12.5 MB. Figures 6 and 10 provide a view of each respective test environment.

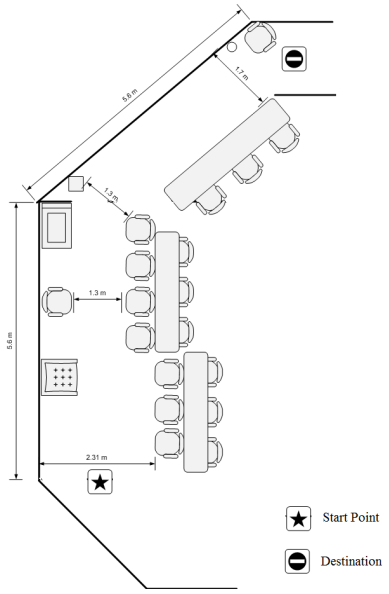
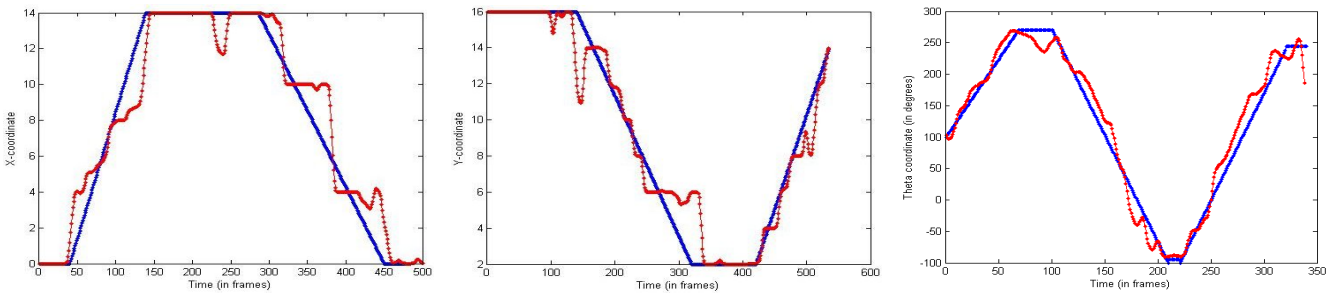
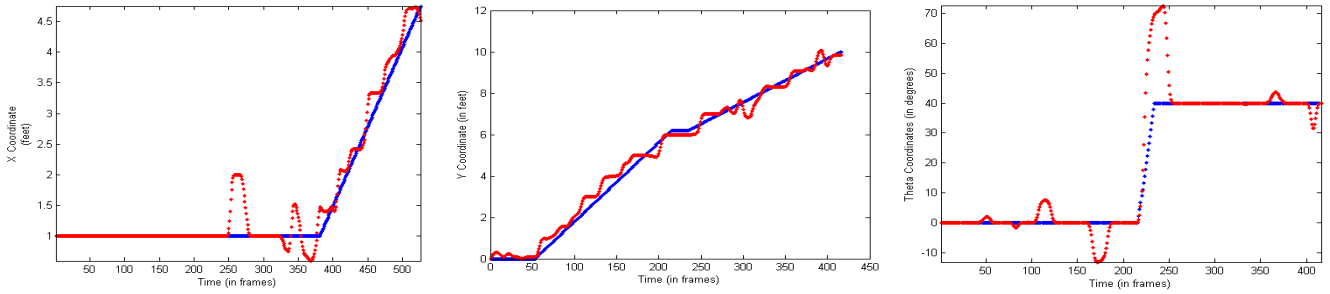


Fig. 7. An overhead view showing the trajectory the helicopter has to follow in E2. Note that this map only shows two degrees of freedom out of a possible four for our helicopter.



(a) x-component (b) y-component (c) θ-component

Fig. 8. A square trajectory in environment E1. Blue shows the ground-truth and red shows the predicted location. (Best viewed in color)



(a) x-component (b) y-component (c) θ-component

Fig. 9. A trajectory in environment E2. Blue shows the ground-truth and red shows the predicted location. (Best viewed in color)

TABLE I

AVERAGE AND MEDIAN ERROR ALONG VARIOUS AXES FOR TWO TEST ENVIRONMENTS (E1 AND E2). "RAND" MEANS THAT WE RANDOMLY PREDICT THE LOCATION/ORIENTATION DRAWN UNIFORMLY FROM POSSIBLE LOCATIONS.

	X (METERS) (MEAN)	X (METERS) (MEDIAN)	Y (ERRORS) (MEAN)	Y (METERS) (MEDIAN)	θ (MEAN)	θ (MEDIAN)
E1 (RAND)	1.22	1.22	1.22	1.22	90.0°	90.0°
E1 (OUR METHOD)	.32	.11	.22	.09	15.6°	12.8°
E2 (RAND)	.46	.46	.91	.91	90.0°	90.0°
E2 (OUR METHOD)	.11	0.01	.36	.29	8.3°	4.3°

A. *Simulator Results*

We collected six test videos in E1 and four in E2. The test videos were collected along various trajectories. Fig. 8 a,b shows the x and y classification results on a typical test video in E1; this video was taken along a square-path where the helicopter was pointing along the direction of movement. Fig. 8c shows another trajectory in which x and y are kept constant, but θ is panned.

We tested our framework on a significantly harder trajectory in E2 which consists of a narrow corridor with obstacles on either sides followed by a sharp turn around a corner and into a second section of the corridor (see Fig. 7). Fig. 9 shows the simulated results of this trajectory. The constrained nature of this path makes accurate classification especially important for robust traversal. We see that the predicted results follow the ground truth trajectory quite closely. As shown in Fig. 7 and 10, the trajectory consists of two laps. In the first part, x and θ remain mostly constant and y increases as we move forward in a nearly straight line while maintaining a safe distance from the sides. In the second part, the helicopter turns (thus changing θ to a different value) and follows the second corridor in which both x and y are increasing.

Table I shows the average and median error for x, y and θ. We get good results for predicting the position and orientation estimates in both environments— for instance, a mean error of only 12.9° (with random baseline of 90°)[5] for the θ estimate. Because of a few errors in the prediction, our mean errors increase significantly. Its is also worth noting that the classification error is low even in E1, despite the environment being virtually featureless. Additionally, our controller can handle a few misclassifications without jeopardizing the flight because it would correct itself in consecutive frames.[6] Therefore, we also report median errors.

These trajectories show typical results and are a reasonable test case for our classification algorithm; however, in reality

[5]Random means that we randomly predict the orientation drawn uniformly from all possible locations

[6]Furthermore, the classifier only provides a global location estimate; local position is well stabilized by the optical flow.

the results can vary due to imperfect modeling of helicopter dynamics. Therefore, we test our algorithm with actual flights in the two test locations.

B. Real Test Flights: System Details

Real-time perception, processing and response is crucial for our algorithm. We therefore performed several optimizations to meet these constraints. Table II shows the execution time of key components of our system when using different dimensions for the PCA projection. The data was collected on a Dell XPS 1330 laptop with a 2.4 GHz Core 2 Duo processor. The on-board video camera captures at 30 frames per second (fps). Using the 32 dimension PCA model in our experiments, we were able to to achieve nearly real-time response by dropping every other frame. Fast response is extremely important for flying the helicopter successfully; therefore, we trade-off a slight degradation in classification accuracy for an order of magnitude improvement in processing time.

TABLE II
COMPARISON OF PERFORMANCE WITH DIFFERENT PCA DIMENSIONS.

	PCA (MS)	KNN (MS)	TOTAL FPS
32-DIM	23.77	4.77	14.32
64-DIM	48.6	7.00	10.10
256-DIM	186.9	487.26	1.40

C. Real Test Flights: Results and Discussion

We performed several experiments in the two environments along various trajectories.[7] We ran experiments using both the Blade CX2 and the CX3 helicopters with no changes to the algorithm. We have the videos available at (some of the clips are also shown in the video attached with the submission):

http://ai.stanford.edu/∼saip/helicopter

We used our algorithm to perform several hovering and panning trajectories. Our algorithm is robust in maintaining orientation. In E1, the open space is nearly symmetric, so several locations (e.g. the hallways) have unusually high rates of misclassification. Despite this, our algorithm is still able to autonomously pan to a desired orientation and maintain stable hovering; the combined update equation allows it to pass through less stable positions until it fixes on the desired location.

Next, we perform a number of trajectories in different environments - for example, an L-shaped trajectory in E1 (not shown in the video). In one trajectory in E2 (see Fig. 10), the helicopter has to fly from one end of the room to another back and forth (including turning automatically) without running into the walls or the chairs. This is a significant challenge in this narrow area, where even a small drift or gust of turbulence can cause the helicopter to hit a wall very quickly. In this case, our algorithm corrects for both lateral and panning drift to avoid nearby obstacles. When the helicopter reaches the end of the corridor, it successfully turns and traverses the corridor again.

In the more challenging trajectory in E2 (shown in Fig. 7 and 10), the helicopter flies through the corridor and autonomously turns around a corner to go to the second part of the corridor. This trajectory was even more constrained than the first. The helicopter had to pass through narrow areas (only 1.30m wide) multiple times—with the diameter of the helicopter (and the landing gear) as 0.36m, this left only 0.47m free space on each side of the helicopter. In this small space, the helicopter must automatically turn around the corner while continuing to maintain a safe distance from the walls and the obstacles. Our attached videos shows a few typical clips out of the many experiments we did in this location.

There are a few limitations of the current system. The throttle controls are very sensitive and we found it hard to automatically maintain the desired height accurately. This causes the helicopter to drift up and down, potentially resulting in a few additional misclassifications. We plan to improve this to fly the helicopter more robustly.

VII. CONCLUSION

We presented a control algorithm for indoor helicopter navigation based on fast nearest neighbors classification for 3D localization and optical flow for velocity estimation. Our algorithm was successfully able to fly a miniature RC helicopter autonomously in known but constrained indoor spaces. The algorithm requires only a single light-weight camera for visual perception and control. Our approach extends the approaches of past work (e.g. 2D ground and pattern-based image localization) to the more difficult problem of 3D flight with a miniature helicopter in multiple challenging environments. We believe that our results demonstrate that in the future, even miniature aerial robots could fly autonomously using only a single camera as the sensor.

[7]Test flight procedure: Landing and taking-off of the helicopter was done manually. After taking-off, the control was handed over to our vision algorithm. The sub-trims of the helicopter control were also set manually before the flight—a standard procedure for human helicopter pilots.

REFERENCES

[1] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Neural Information Processing Systems (NIPS)*, 2006.

[2] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *International Conference on Machine Learning (ICML)*, 2006.

[3] S. Bouabdallah and R. Siegwart. Backstepping and sliding-mode techniques applied to an indoor micro quadrotor. In *International Conference on Robotics and Automation (ICRA)*, 2005.

[4] J. Bouguet. Pyramidal implementation of the lucas kanade feature tracker description of the algorithm. OpenCV, Intel Corporation, 1999.

[5] A. Coates, P. Abbeel, and A. Y. Ng. Learning for control from multiple demonstrations. In *International Conference on Machine Learning (ICML)*, 2008.

[6] M. Cummins and P. Newmann. Fab-map: Probabilistic localisation and mapping in the space of appearance. *International Journal of Robotics Research (IJRR)*, 27(6):647–665, 2008.

Fig. 10. Sequence of images showing our helicopter flying and turning the corner to complete the trajectory in E2.

[7] X. Deng, L. Schenato, W.-C. Wu, and S. Sastry. Flapping flight for biomimetic robotic insects: Part ii- flight control design. *IEEE Trans on Robotics*, 22(4):789–803, 2006.

[8] E. Feron and S. Bayraktar. Aggressive landing maneuvers for unmanned aerial vehicles. In *AIAA Guid, Nav Cont Conf*, 2006.

[9] V. Gavrilets, I. Martinos, B. Mettler, and E. Feron. Control logic for automated aerobatic flight of miniature helicopter. In *AIAA Guid, Nav Cont Conf*, 2002.

[10] R. He, S. Prentice, and N. Roy. Planning in information space for a quadrotor helicopter in a gps-denied environments. In *International Conference on Robotics and Automation (ICRA)*, 2008.

[11] Hokuyu. Range-finder type laser scanner urg-04lx specifications. Online, 2005.

[12] N. Johnson. *Vision-Assisted Control of a Hovering Air Vehicle in an Indoor Setting*. PhD thesis, Bringham Young University, 2008.

[13] E. Klingbeil, A. Saxena, and A. Y. Ng. Learning to open new doors. In *Robotic Science and Systems (RSS) workshop on Robot manipulation*, 2008.

[14] L. Mejias, J. Roberts, K. Usher, P. Corke, and P. Campoy. Two seconds to touchdown vision-based controlled forced landing. In *Int'l conf on Intelligent Robots and Systems (IROS)*, 2006.

[15] J. Michels, A. Saxena, and A. Y. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2005.

[16] R. Mori, K. Hirata, and T. Kinoshita. Vision-based guidance control of a small-scale unmanned helicopter. In *Int'l conf on Intelligent Robots and Systems (IROS)*, 2007.

[17] J.-D. Nicoud and J.-C. Zufferey. Toward indoor flying robots. In *Int'l conf on Intelligent Robots and Systems (IROS)*, 2002.

[18] R. Panigrahy. An improved algorithm finding nearest neighbor using kd-trees. *LNCS*, (4957):387–398, 2008.

[19] E. Ribnick, S. Atev, and N. Papanikolopoulos. Estimating 3d positions and velocities of projectiles from monocular views. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 31(5):938–944, 2008.

[20] J. Roberts, T. Stirling, J.-C. Zufferey, and D. Floreano. Quadrotor using minimal sensing for autonomous indoor flight. In *European Micro Air Vehicle Conference (AV)*, 2007.

[21] D. Schafroth, S. Bouabdallah, C. Bermes, and R. Siegwart. From the test benches to the first prototype of the mufly micro helicopter. *J Intell Robot Syst*, 54:245–260, 2009.

[22] B. Steder, G. Grisetti, C. Stachniss, and W. Burgard. Visual slam for flying vehicles. *IEEE Transactions on Robotics*, 24(5):1088–1093, 2008.

[23] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

[24] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: a large dataset for non-parametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

[25] G. Tournier, M. Valenti, and J. P. How. Estimation and control of a quadrotor vehicle using monocular vision and moirre patterns. In *AIAA Guidance, Navigation, and Control Conf*, 2006.

[26] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuriscience*, COM:132, 1990.

[27] B. Williams, M. Cummins, J. Neira, P. Newmann, I. Reid, and J. Tardos. An image-to-map loop closing method for monocular slam. In *Int'l conf on Intelligent Robots and Systems (IROS)*, 2008.