Vision-Controlled Autonomous Indoor Helicopter

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

Thanks to Ashutosh Saxena (asaxena@stanford.edu)

Abstract- **Indoor navigation of aerial vehicles poses several challenges in sustaining hover flights. The key challenges are the high presence of obstacles in constrained environments, frugal payload budget and the need for real-time control response. In this paper, we describe a successful application of computer vision and machine learning techniques for autonomous panning of RC helicopters in fixed indoor settings. Our framework uses images of the environment to 'learn' its surrounding and perform attitude estimation on live-streaming data from an on-board wireless camera using the learned model. We then detail the design of our controller, capable of hovering the helicopter in place, sustaining orientation with respect to reference points and following panning trajectories. Finally, we present test results based on simulations and actual autonomous test flights.**

I. Introduction

Unmanned aerial vehicles have several indispensable applications in surveillance, intelligence and information relay. Small radio controlled helicopters are particularly suited for indoor navigation owing to their stable low-speed flight and in-place hovering capabilities. The restricted nature of indoor environments with walls and other obstacles, however, pose unique challenges for autonomous navigation and necessitate real-time control response. Frugal payload budget and power restrictions of RC helicopters place additional constraints on the use of on-board sensors. These make indoor autonomous navigation an interesting problem, from both the vision and control aspect. We use monocular vision and machine learning techniques to extract information from images streamed from a light-weight wireless camera mounted on the helicopter and transmit back appropriate control signals for autonomous panning.

Several related work on autonomous helicopter flight in outdoor settings [1], [2] have led to terrific control algorithms and are capable of performing extreme acrobatics. Our challenges are quite complimentary to these:

- Indoor navigation is more of a perception and navigation problem than a control problem: it is crucial to be able to detect and avoid obstacles at close quarters in real-time.
- The smaller RC helicopters are limited in their payload capacities and suffer from air turbulence generated by its own motion in closed spaces
- Hazy images from the on-board video capture are subject to severe vibrations from helicopter motion.

To address these issues, our framework is composed of two tightly coupled systems: the vision and the control system. During the training phase, the framework forms a model of the environment based on labeled images of its surrounding. This model can be used to specify motion paths for flight, such as stable hovering with respect to reference points or simple panning trajectories. The vision system performs attitude and motion estimation on images streamed from the onboard camera. The control system uses these estimates to generate panning control signals and interfaces with the helicopter's transmitter to efficiently control navigation for sustained flight. Fig. 1 shows panning direction in a 3D space.

II. Helicopter Platform

Our test platform is based on the EFlite Blade CX2 coaxial helicopter (Fig. 2). We started working with a dual-rotor model (Exceed RC), but faced severe instability and lateral drift issues. The coaxial model was chosen over this since they provide stable flight and are considerably easier to control than their dual rotors counterparts. A light-weight wireless camera is mounted on-board (Fig. 1 inset) to stream images in real-time to the control algorithm. We use a commercial interface, called the PCTx [3], to link the transmitter to the USB port of the laptop, via the trainer port (Fig. 3). This enables the helicopter to be controlled via the PC using the control system of our framework. We encountered several compatibility issues in getting this to work since the transmitter (Tx) shipped with Blade CX2 was unable to accept

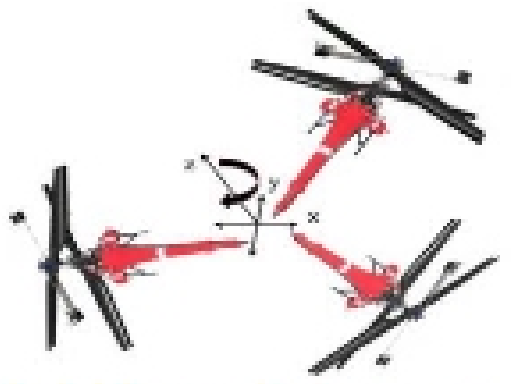

Fig. 1. Diagram of helicopter panning

Fig. 2. The Blade CX2 Helicopter

Fig. 3. The platform interfaces

PPM signals from the trainer port, which is what PCTx uses to communicate with the Tx. We now use the Spektrum DX6i Tx, which is capable of translating PPM input to DSM signals for controlling the helicopter.

III. CONTROLLER FRAMEWORK

We use techniques from computer vision and machine learning to be able to quickly and reliably get an estimate of our current position and velocity. We chose the K-Nearest Neighbors (kNN) algorithm to classify a frame with respect to its attitude because it is a fast, simple, data-driven approach. However, this alone is insufficient for fine-grained navigation and stability. We calculate optical flow to measure the relative velocity from frame to frame. Our goal is to combine these estimates to produce sensitive control signals capable of sustaining autonomous panning flight. Fig. 5 shows the block diagram for the navigation system. The right side shows the control path for live helicopter flight, while the left side shows the control flow for the virtual controller, discussed in section IV.

A. Training the Model

In the training phase we attempt to learn a model of the environment. The choice of a constrained setting is an important factor; an environment that is too dynamic (varying from day to day tests) will perform poorly. We began with a training set taken along an oval in the AI Lab. A training video is taken at each point by one of us holding the helicopter facing a fixed reference point and rotating 360 degrees. Each frame in a video is then labeled using its angle w.r.t. a reference point.

$$Label = \frac{Current\ Frame\ Number}{Total\ Frames} * 360$$

This procedure has some weaknesses. First, data collection can be a slow process if there are many points in the environment. Second, it is difficult for a human to rotate at a uniform velocity. Lastly, the AI lab has many students and many gadgets in a state of flux. To address some of these problems, in the second half of the quarter we moved our training environment to the 4th floor of Gates, just outside of the elevators (Fig. 4).

Fig. 4. The training environment

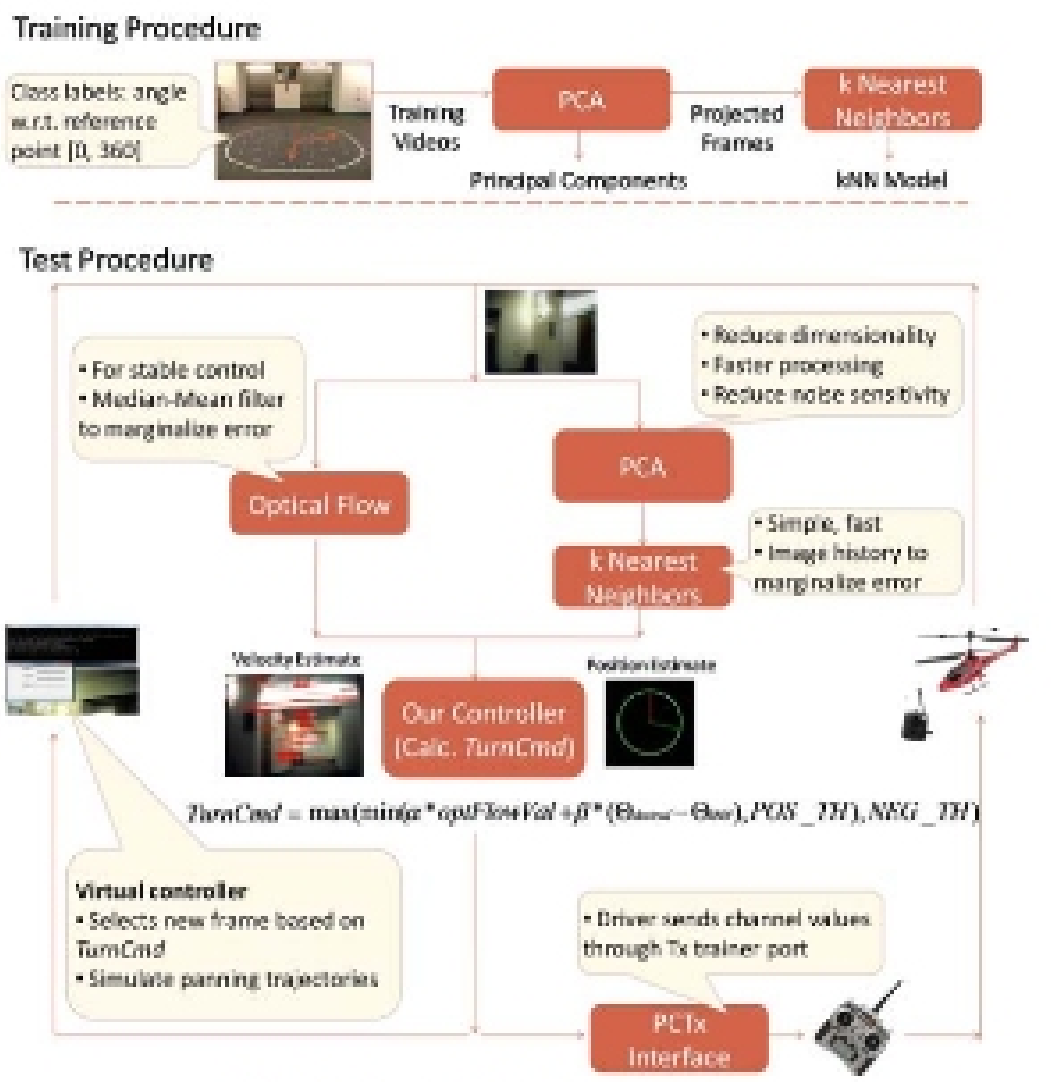

Fig. 5. Control flow diagram

This is an open space suitable for initial testing that does not change much from day to day. We collected two sets of training data along and within circles of varying radii, each at a different height. For each point, we took a video rotating first in the clockwise direction and then in the counter-clockwise direction; we hypothesized that this would help marginalize some of the human error involved from point to point. We also experimented with methods of removing human action from the process by rotating the helicopter on a turntable, but found that we were unable to slow the rotation to a satisfactory speed.

Our final training dataset consisted of 70 videos and 16089 frames (each frame is 640 x 480). As this is a large amount of data, we originally resized each frame to 50x50 to meet memory and speed constraints for real-time processing. Eventually, we found that we could improve our results by using Principal Components Analysis (PCA) to reduce each frame's dimensionality from 307200 to 10. PCA was implemented using OpenCV's computationally efficient built-in eigen objects and eigen projections. Instead of arbitrarily re-scaling the image, using PCA also helped us reduce the classification's sensitivity to noise perturbations and height variations. Finally, we trained the K Nearest Neighbor's model on the projected data using OpenCV's built-in KNN implementation.

B. Attitude and Velocity Estimation

When a new frame is received from the helicopter's wireless camera, we extract the K nearest neighbors from its PCA projection. We incorporate the image history on the last five

Fig. 6. Visualization of optical flow with attitude inset

classifications to select the closest prediction (i.e. predictions of the previous five frames are used to find the closest match from among the K results), and that is output as the prediction of current attitude.

Optical flow is applied on the original full-sized frame to give us a measure of which direction the helicopter is moving in. This velocity estimate will be fed into the control algorithm and used to refine the magnitude of the turn command generated. This provides a counter-acting effect that helps prevent the helicopter from turning too drastically. Optical flow is implemented using 400 features and OpenCV's Pyramidal Lucas Kanade (PLK) algorithm [4]. We calculate the total optical flow as the mean over all the features for each frame, with respect to the horizontal direction only. Finally, we apply a median-mean filter to smooth the optical flow values over time. Fig. 6 shows the calculated optical flow on a frame, and the inset shows the attitude estimated by kNN.

B. *Control Algorithm*

We propose an algorithm to combine the optical flow result with the K nearest neighbor classification to produce an updated attitude estimate for each frame. The update equation is given as follows:

$$TurnCmd = \max(\min(\alpha \, \emptyset_{optical} + \beta(\theta_{desired} - \theta_{kNN}), P\_TH), N\_TH)$$

where $\emptyset_{optical}$ is the velocity estimate calculated by optical flow, $\theta_{desired}$ is the desired angle we wish to navigate to, θ_{kNN} is our current estimate from kNN. α and β are constants used to account for time and tune the result. We can alter $\theta_{desired}$ as a function of time to make the helicopter follow a particular trajectory at fixed altitude. *TurnCmd* is the control signal generated by the algorithm, which is passed on to the transmitter to autonomously control the helicopter panning. The *TurnCmd* value is clipped to lie between the negative and positive threshold, N_TH and P_TH, to avoid over-controlling.

IV. Controller Interface and Simulator

A. *Controller Interface*

We have developed a functional GUI framework for remotely controlling the RC helicopter. Our framework is built on top of the OpenCV library and Endurance's low level PCTx driver. It was designed to provide us flexibility in running experiments, smooth control of navigation and effective real-time feedback. It also serves as the UI for training and running the simulator.

The controller maps the transmitter controls to intuitive keyboard presses, allowing the user to easily control the helicopter in real-time. It is multi-threaded with non-blocking I/O to ensure responsiveness. With a single key press, the user can transfer control of the rudder (for panning) to the already running Vision Control algorithm, while continuing to control the throttle, aileron, and elevator from the keyboard. To prevent the helicopter from receiving wild swings in control signals which could damage its gears, the controller has a configurable maximum change per channel per time step. The frequency of updated channel values being transmitted to the helicopter is also configurable from the interface. Fig. 7 shows a screenshot of the controller during an actual flight. The display presents several pertinent real-time results to the user: the current desired and actual channel values, the video stream from helicopter's wireless camera, a radar displaying the current desired panning angle and actual kNN estimate, and trackbars showing the kNN estimate, optical flow values, and turn command generated by the control algorithm.

B. *Simulator*

We implemented a simple simulator, or virtual controller, to verify our framework. The virtual controller forms a map of the surrounding based on the test videos. It interacts with the vision and control framework by accepting a turn control command, retrieving the image frame that would have resulted if the command was issued to the helicopter in the actual setting, and passing that on to the controller. This allows for simulating panning trajectories and visualizing the controller's dynamics. The simulator was used extensively in the testing of our system framework.

Fig. 7. Screenshot of the controller interface