Quadcopter Path Planning Using Optical Flow

Drew Mitchner

MEng Report - Fall 2018

1 Introduction

The RoboBee, a flapping microrobot, can sustain hover and follow trajectories using an adaptive spiking neural net controller. Due to power requirements and to mimic actual insect flight, a neuromorphic camera allows the RoboBee to interact with its environment and allows application of path-planning algorithms. However, hardware constraints including lack of appropriately sized neuromorphic sensor and lack of on-board power or computing prevents current tests of control algorithms. Thus, two methods of testing control algorithms are proposed and developed.

First, a hardware-in-the-loop configuration will simulate the neuromorphic camera in conjunction with Blender, a modeling software used to create photorealistic images and animations. The simulated image will be fed into the controller, which outputs the desired control inputs for the robot.

Second, a micro-aerial vehicle equipped with RGB-camera is developed which can execute and test the same control algorithms which will be used on the RoboBee.

2 Hardware

2.1 RoboBee

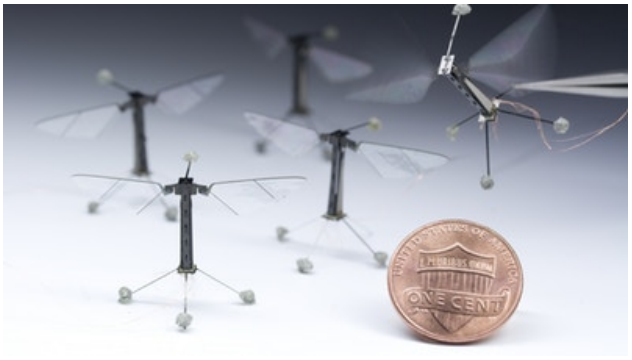

Figure 1

The RoboBee is developed at the Wyss Institute at Harvard University. Leveraging new pop-up MEMS technology, the RoboBee achieves micro-scale flight by flapping its wings up to 120 times a second.

2.2 CrazyFlie 2.0

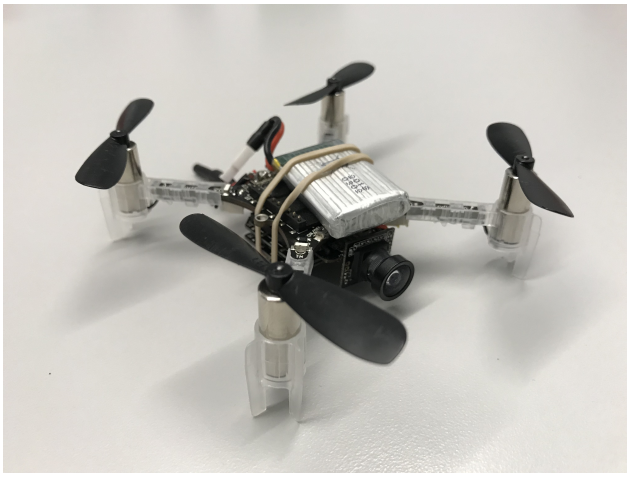

Figure 2

The CrazyFlie 2.0 is a micro aerial quadrotor developed by BitCraze. It has on-board IMU and optic flow sensor for hover-assist, as well as a Python API interface.

2.3 Camera Deck

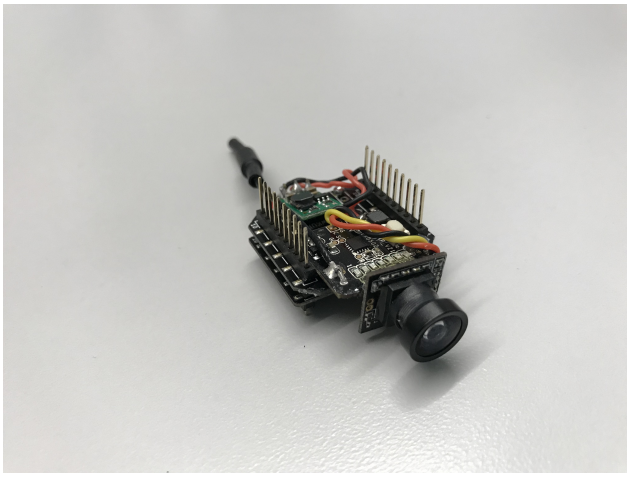

Figure 3

The camera deck is comprised of a Turbowing DVR Cyclops FPV Camera, 5V step-up regulator, and transceiver. The regulator provides a steady, 5V input to the camera, which takes RGB video in real time, and the transceiver sends the data to a off-board computer.

3 Hardware-In-The-Loop Configuration

The hardware-in-the-loop simulation allows the RoboBee to execute control inputs without sensors or on-board computing. An overview of the configuration is shown below.

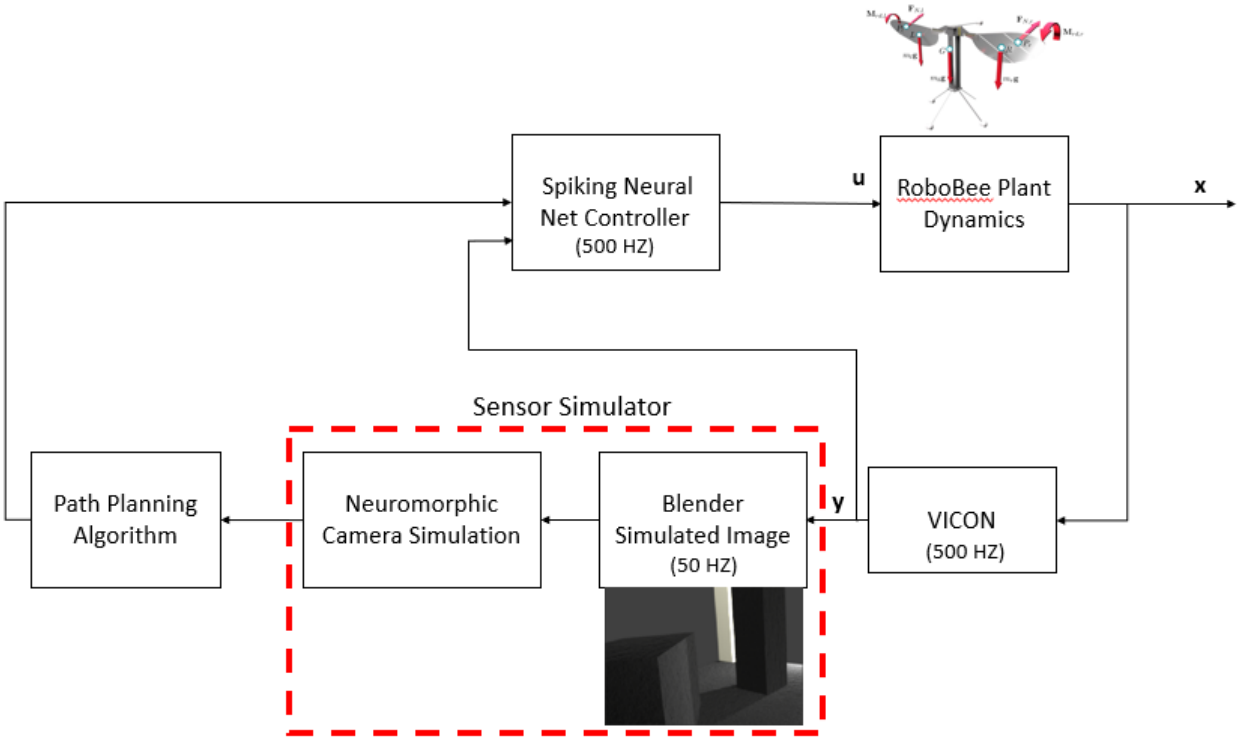

Figure 4: Overview of the control loop architecture used for the hardware-in-the-loop configuration.

In this configuration, the VICON motion capture system detects the robot state. A script then takes the most recent state measurement and moves the camera inside a Blender API and renders an image of what the robot would see if it did have its on-board sensors. The image is then converted into a neuromorphic image and is the resulting data is used in creating high-level control inputs.

To test the rendering speed in Blender, a Python script runs through 1,000 different camera locations and renders each image, and returns the pixel data to use for path planning. The results of these tests are shown in the table below.

Setup	Image Rate	Camera Time	Render Time	Pixel Time
OpenGL (with UI)	28 Hz	0.1 ms	21.6 ms	9.8 ms
Blender Render (with UI)	11.5 Hz	0.1 ms	78.5 ms	8.5 ms
Blender Render (without UI)	11.8 Hz	0.01 ms	85.3 ms	0.01 ms

Table 1: Results of the 1000-image tests measuring speed of the sensor simulator. The current setup generates images at about 11.8 Hz.

An ideal goal is to be able to run this sensor simulation at 50Hz, but a slower rate would also be effective.

4 Quadrotor Path Planning

A path-planning demo utilizing the quadrotor will demonstrate the advantages of the neuromorphic sensor and corresponding neuromorphic algorithms over a traditional frame-based camera. The proposed advantages are: reduced power consumption, reduced computation, and quicker response times leading to more robust obstacle avoidance or target tracking.

The algorithms used for path-planning depend heavily on assumptions made for the demonstration. There will be only one moving target, and all other obstacles will be stationary. There are four main aspects to accomplish this task.

1. Target recognition and tracking
2. Obstacle detection and avoidance
3. Path planning
4. Control inputs

4.1 Target Recognition and Tracking

In the demo, the quadrotor will identify and follow a moving target. Because there will only be one moving object in the scene, any movement detected will be assumed to be the target.

To accomplish this, the quadrotor will begin in a "loiter" phase, where it hovers in place. When optical flow values of a region of pixels exceeds a certain threshold, the robot will detect the corner pixels in the region and identify these as the target.

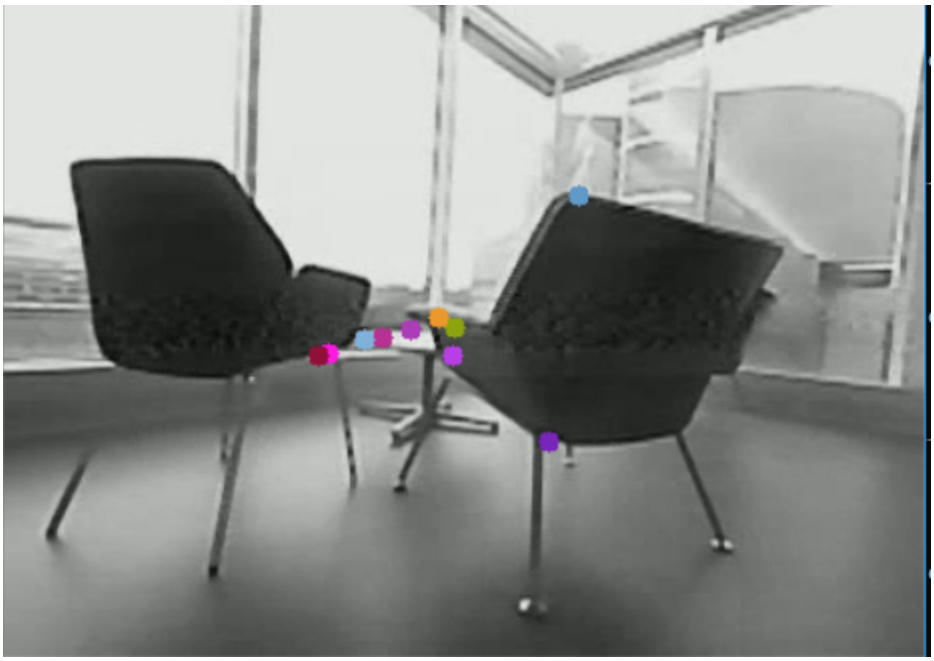

Figure 5: Optical flow used to detect and track pixels of interest in real time.

Visual servoing will be used to keep the target roughly in the center of the frame. The rotational yaw velocity can be used to center the target laterally, with PD-control used to determine the magnitude and direction of the rotation rate. Overall, this yaw rotation is always considered independently from the rest of the motion of the system - the quadrotor always attempts to point at the target.

Visual servoing will also be used to center the target vertically, though the quadrotor will be required to move up or down to achieve this, as pitching forward or back as with laterally will cause the vehicle to translate in an undesirable way.

This method allows the robot to quickly re-detect the target if it loses track of a certain percentage of the target pixels. The robot re-enters the "loiter" phase, and quickly identifies the moving object.

4.2 Obstacle Avoidance

As with the rest of the path-planning, obstacle avoidance will utilize the Farneback dense optical flow algorithm to acquire the optical flow of every pixel at every time step. A visualization of this algorithm is shown below.

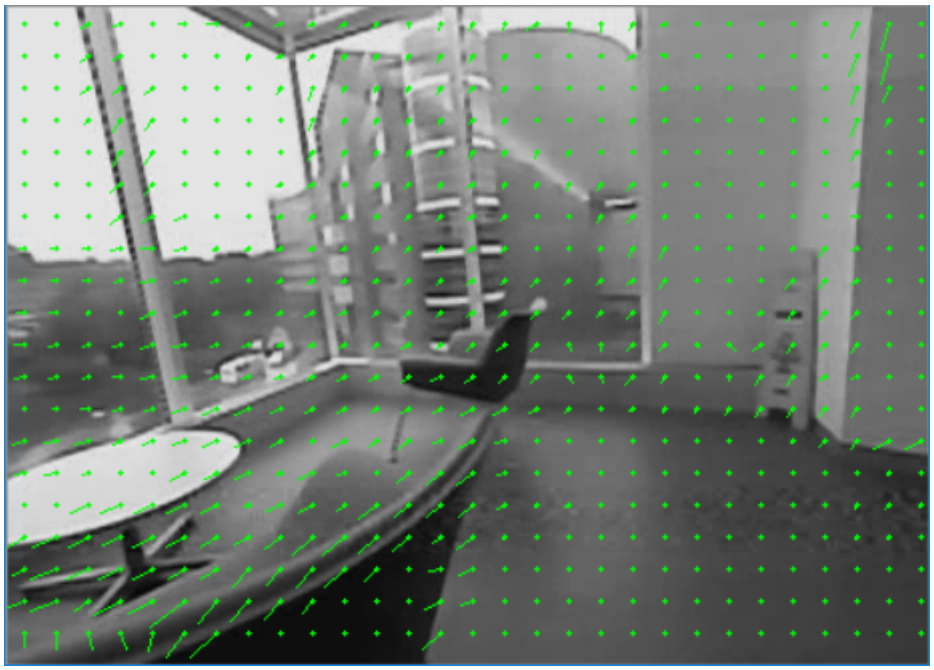

Figure 6: Farneback method calculates optical flow at all points.

The magnitude of the optical flow at each point is related to the distance away an obstacle is (larger magnitudes indicate closer obstacles). By using Structure-From-Motion, the quadrotor can leverage the stationary obstacles assumption and use its single camera as if it were stereo vision by combining two proceeding frames. This allows the vehicle to estimate the depth of obstacles. By using an inverse square law or Gaussian curve, the depth to each pixel is translated into a force on the quadcopter, and the sum of the total force results in a net repulsive force on the vehicle.

4.3 Path Planning

The overall path planning structure is thus similar to a potential field algorithm. The target pixels create an attractive potential, which influences the translational and rotational velocities of the vehicle. The optical flow of the obstacles creates a repulsive potential which solely influences the translational motion.

Manual tuning of the relative magnitudes of these effects will be required to ensure they are all of similar magnitude.

4.4 Control Inputs

Control inputs will be sent to the vehicle via the CrazyFlie Python API. Specifically, the function "_set_vel_setpoint(velocity_x, velocity_y, velocity_z, rate_yaw)" will be used to set each of the body frame velocities and yaw rate at each time step.