Nanocopter Project Final Report

COSC 50 – Winter 2014

The Capitalists – Channel 26 – SVN raveyard version 5736
Charles Dong
Jamie Mitchell

I. Project Description

The nanocopter project involved the design and implementation of a system that controls an embedded device in real-time by a small team. Specifically, this project involved the translation of hand gestures read by a Leap motion sensor into radio signals that manipulated a Crazyflie quadcopter drone.

The motion sensor receives a steady stream of gestures, or commands, from the user. The seven distinctive gestures we designed were forward, backward, left, right, up, down, and swipe. We also allow the user to draw out a pattern with a "pointable object" such as a pencil, which the copter will record and then fly in hover mode (a flight mode in which the nanocopter maintains the altitude at which it entered hover mode).

Depending on the "state" of the copter, the real-time control system sends the appropriate roll, thrust, pitch, and yaw commands to the nanocopter, which correspond to flying right, left, forward, backward, up, and down.

II. User Instructions

Basic instructions: forward, backward, left, right, up, and down hand gestures make the nanocopter fly in the appropriate direction. Swipe gestures will cause the nanocopter to enable/disable hover mode, in which it maintains constant altitude.

Project extension: user can use a long, thin object (such as a pencil) to draw a pattern (forward, backward, left, and right). The pattern will be recorded and flown by the copter. Note that the copter will maintain altitude in hover mode, so up/down motions are not recognized.

Hand gestures (with straight, flat hand):

- Forward: tilt hand >45° forward (fingers down, wrist up)
- Backwards: tilt hand >45° backwards (fingers up, wrist down)
- Left: tilt hand >45° left (for right hand: thumb down, pinky up)
- Right: tilt hand >45° right (for right hand: thumb up, pinky down)
- Up: lift hand up further above Leap motion sensor
- Down: move hand down toward Leap motion sensor

Extension gestures:

- With one hand, use a long, skinny object such as a pencil to draw a pattern. The tip of the object will be read by the Leap motion sensor.
    - This is the recording stage, during which the nanocopter will hover in place automatically
    - Note that the nanocopter will ignore the up and down portion of user patterns

- When done drawing the pattern, put your other hand into view of the Leap motion sensor (in addition to the first hand holding the object)
    - At this point, the nanocopter will enter its pattern-flying stage and fly the pattern the user drew in hover mode.
    - When it is finished flying the pattern, it will go back into whatever mode it was in before the user began recording a pattern.

III. Threaded Design

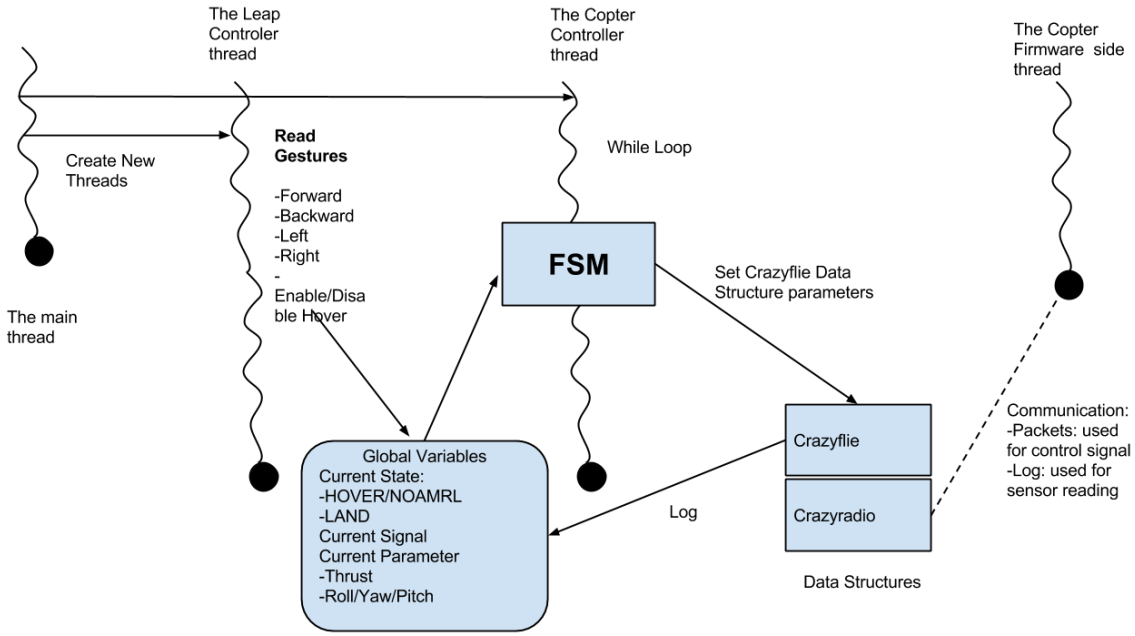

Figure: Threaded design diagram from CS50 Design Document

Leap Controller Thread

The leap controller thread continually reads and recognizes hand gestures from the Leap motion sensor through callbacks (most importantly, the infinite loop calls the function on_frame continually whenever the user's hand is seen by the motion sensor). This thread then translates gestures into various "states" and appropriate roll, pitch, yaw, and thrust numbers to be sent to the nanocopter. Specifically, it writes to the following global variables, which we will discuss the purpose of in the Finite State Machine Design section:

```
extern unsigned short current_thrust;
extern float current_pitch;
extern float current_roll;
extern float current_yaw;
extern int current_state;
extern int current_signal;
extern int8_t flightModeHover;
```

extern int FLY_PATTERN;

Using the Leap Motion API, the system reads the user's hand position within the 3-dimensional subspace above the leap motion sensor and stores each recognized hand as x, y, and z coordinates – by taking the difference between frames, it also calculates velocity. Thus the system can recognize our defined gestures through the interplay between the velocity, direction, position, etc. of the user's hands.

jamieOnFrame function pseudocode

This function is called within the Onframe function every time the Leap Motion detects a hand in its field of view. This function is responsible for deciphering the leap motion frame information into hand gestures. The function uses the following frame information to determine gestures:

- Number of hands
- Number of fingers
- Hand position vector
- Hand direction vector
- Hand velocity vector
- Hand normal vectors
- Presence of pointable tool object

If the hand is determined to be stationary and tilted past the threshold value, the values of pitch, roll, and thrust are decreased or increased based on a constant value. Similarly, if the criterion for a gesture is detected, the current signal is updated. Note that gestures are arranged such that no two gestures could possibly be detected within the same jamieOnFrame call.

The function does not return any values, but writes to multiple global variables per call. The function will always update pitch, roll, and thrust but based on the current state, these values may be overwritten before being sent to the Nanocopter.

Nanocopter Controller Thread

The nanocopter controller thread continually reads from the global variables and, depending on the current state, sends CCRTPPackets containing appropriate thrust, pitch, roll, yaw, and hover mode enable/disable commands to the Crazyflie nanocopter through the Crazyradio. It also reads the battery level log information from the nanocopter.

Below, please find a brief overview of the abovementioned control parameters pitch/yaw/roll/thrust. Note that our nanocopter treats the vector from the M3 motor to M1 motor as forward.

- Pitch: moves nanocopter forwards or backwards
- Roll: moves nanocopter left and right
- Yaw: change the nanocopter's heading direction left and right
- Thrust: moves nanocopter up and down, a larger thrust value will increase lift and thus altitude

The data structures used in the real-time control system consist of the CCrazyflie copter (which holds the all-important roll/pitch/yaw/thrust instance variables to be sent to the nanocopter) CCrazyradio (which is connected to the nanocopter's channel through a USB dongle device) and global variables listed above.

IV. Finite State Machine Design

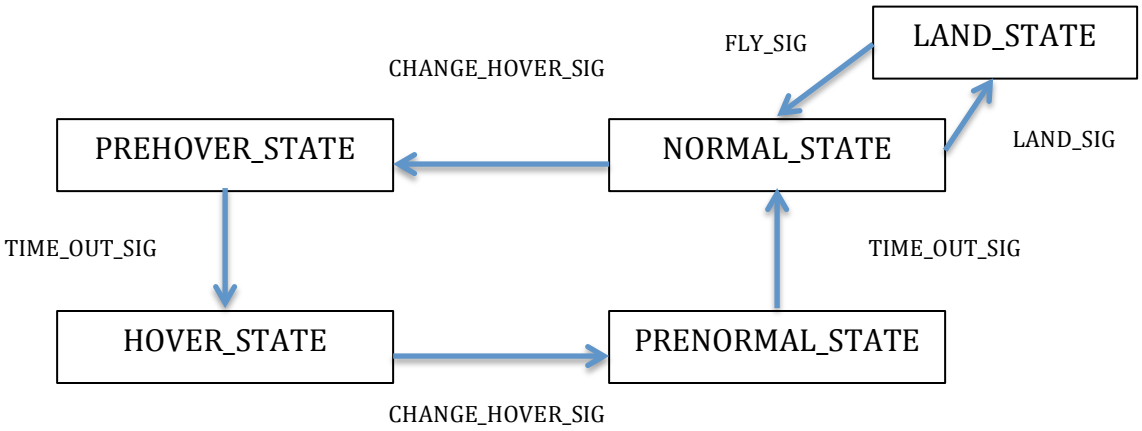

Figure: Finite State Machine representation – squares represent states, and the text in between states represents the signals to switch into said state

Above, we have illustrated our finite state machine design pictorially. The text boxes represent the five states, and and the *_SIG represent the signals issued by the threads to switch between states. Note that the arrows represent the direction in which states can change when given the appropriate signal.

- LAND_STATE: this state doesn't do anything except wait for a FLY_SIG.
  - FLY_SIG from user (two hands with all five fingers extended) will cause the nanocopter to lift off and enter NORMAL_STATE
- NORMAL_STATE: in this state, the user can move the nanocopter up, down, left, right, forward, and backwards. The user can also record/fly a pattern in this state.
  - LAND_SIG from user (two hands with two fingers extended by each hand) will cause the nanocopter to slowly decrease thrust to 0 and land the nanocopter

    - CHANGE_HOVER_SIG from user (swipe gesture) will cause the nanocopter to enter PREHOVER_STATE
- PREHOVER_STATE: transition state between NORMAL_STATE and HOVER_STATE in which no user gestures are accepted – the nanocopter will hover in place at whichever altitude it was in when the user swiped until it fully transitions into the HOVER_STATE.
    - After a set amount of time, a TIME_OUT_SIG signal will be issued to transition into the HOVER_STATE
- HOVER_STATE: in this state, the user can move the nanocopter left, right, forward, and backwards. The nanocopter will maintain the same altitude as in PREHOVER_STATE. The user can also record/fly a pattern in this state.
    - CHANGE_HOVER_SIG from user (swipe gesture) will cause the nanocopter to enter PRENORMAL_STATE
    - LAND_SIG from user (two hands with two fingers extended by each hand) will cause the nanocopter to slowly decrease thrust to 0 and land the nanocopter
- PRENORMAL_STATE: transition state between HOVER_STATE and NORMAL _STATE in which no user gestures are accepted – the nanocopter will hover in place at the same altitude it was at in HOVER_STATE until it fully transitions into the NORMAL_STATE.
    - After a set amount of time, a TIME_OUT_SIG signal will be issued to transition into the NORMAL_STATE

<u>Main Nanocopter Control Implementation</u>

This function contains an infinite while loop that does the following at every iteration:

1. Sends the CCrazyflie object's roll/yaw/thrust/pitch and hover mode enable/disable to the nanocopter
2. Checks the current signal and does the following depending on the signal:
    a. No signal
        i. If we are in NORMAL_STATE or HOVER_STATE:
            1. If we should be flying a pattern
                a. If the next frame of the pattern is not NULL
                    i. Continue flying pattern using next frame
                b. Else notify system to stop flying pattern in the future
            2. Else update the CCrazyflie object's roll/yaw/thrust/pitch and hover mode enable/disable with the current global variables (which are continually set by the leap thread)
    b. Land signal
        i. If the current state is not LAND_STATE, land the copter
    c. Change hover signal

i. If we are in NORMAL_STATE
    1. Change to PREHOVER_STATE
    2. Tell the nanocopter to enable hover mode
    3. Call countTimeOut();
ii. If we are in HOVER_STATE
    1. Change to PRENORMAL_STATE
    2. Call countTimeOut();

d. Time out signal (issued by countTimeOut)
    i. If we are in PREHOVER_STATE
        1. Change to HOVER_STATE
    ii. If we are in PRENORMAL_STATE
        1. Change to NORMAL_STATE
        2. Disable hover mode
e. Fly signal
    i. If we are in LAND_STATE
        1. Lift off and change to NORMAL_STATE

V. Implementation of minor functions

The pseudocode for the leap and nanocopter control threads illustrated above do the heavy lifting but are supported by myriad support functions, which are broken down by category below.

Leap-Specific Functions

n/a – all of the work is done in on_frame and jamieOnFrame (pseudocode above)

Basic Nanocopter Functions

land

Description: decrease thrust to zero
Input: pointer to copter
Return: n/a

** Pseudo Code **
(1) Zero out thrust, pitch, roll
(2) Maintain thrust at default level for constant landing time
(3) While thrust is above zero, decrease thrust by a constant amount.

*****

fly

Description: Sends current pitch and current roll to Nanocopter based on global variables. Sends thrust to Nanocopter if it is not currently in hover mode
Input: Pointer to copter
Return: n/a

** Pseudo Code **

(1) Update Nanocopter's roll, pitch, and yaw
(2) Determine if in hovermode
(3) Update or don't update based on mode

*****

liftoff

Description: Gives the Nanocopter adequate thrust to obtain liftoff
Input: Pointer to copter
Return: n/a

** Pseudo Code **
(1) Zero out current pitch, roll, and yaw
(2) Set thrust to liftoff value
(3) Send values to Nanocopter

*****

charlieSendParam

Description: Enables/disables hover mode;
Input: Pointer to copter, integer althold
Return: Boolean. True if received the hovermode signal, false if not.

** Pseudo Code **
(1) Obtain hovermode variable id from Nanocopter
(2) Create a string buffer
(3) Copy althold value to buffer
(4) Create a packet
(5) Send to correct port on correct channel
(6) Check if packet was received

*****

Nanocopter Functions Used In Extension

new_frame_node

Description: Creates a new FrameNode object to store position x, y, and z
Input: Integer position x ,y ,and z
Return: Pointer to the FrameNode

** Pseudo Code **
(1) Make new frame node
(2) Set instance variables

*****

freeFrameNodes

Description: Deletes all the frameNodes in the DLL starting at the input node
Input: FrameNode to begin deletion process
Return: N/A

** Pseudo Code **
(1) Cycle through all framenodes
(2) Store next frame node
(3) Delete current frame node

*****

record_tool_movement

Description: Creates a FrameNode for the given frame object inside onFrame function and inserts it into the DLL of FrameNodes. Note: X,Y,Z position is recorded in Millimeters from Leap origin.
Input: Tool position vector
Return: Pointer to the FrameNode that it created

** Pseudo Code **
(1) Get X, Y, and Z position from position vector
(2) Make a new frame node with those values
(3) Check if this is the first node after head.
(4) If first node, set the head's instance variables to newly created node's
(5) Otherwise insert it after the current node
(6) Increment the node count and change the current node global to the new node
---
*****

FSM-related functions

countTimeOut

Description: Count a certain amount of time then update the signal to the time out signal. Must also zero out thrust, roll, pitch values and send to Nanocopter
Input: Pointer to copter, current roll, current pitch, current thurst, current yaw, Pointer to current_signal
Return: N/A

** Pseudo Code **
(1) Zero out roll, pitch, thrust, yaw
(2) Send values to Nanocopter. (Makes Nanocopter stationary)
(3) Wait a constant amount of time
(4) Update current signal to time out signal
*****

VI. Lessons learned

The Nanocopter project presented a set of challenges that were not previously encountered in lab work. Mainly, installing the necessary dependencies, understanding and interacting with the given source code, interfacing through Leap motion API, understanding the flight jargon and tendencies of the Nanocopter, and creatively implementing project extensions. It was often difficult to determine whether the source of a given problem was traceable to written code, a Leap motion/radio dongle problem, or a Nanocopter flight/hardware issue. Although we quickly learned of the importance of having a well-balanced Nanocopter, it was much harder in practice to reach equilibrium. Often a propeller would be damaged or the battery would change positions after a crash. In hindsight, attaching training wheels would have helped our team avoid a large bit of Nanocopter adjustment time.

VII. Appendix

Jamie and Charles split up the work for this project evenly (including design, coding, testing, etc.).

Decomposition of code that students wrote/edited:

1) control.cpp – main thread, leap thread, nanocopter control thread
2) charlieControl.cpp – nanocopter-related functions
3) jamieControl.cpp – leap-related functions
4) studentHeader.h – defines global variables and constants
5) CCrazyflie.cpp – functions that send information to the nanocopter

To build and run:

To build: cd into nanoProj/build and type "cmake ..; make"
To run: cd into nanoProj/bin and type "control"