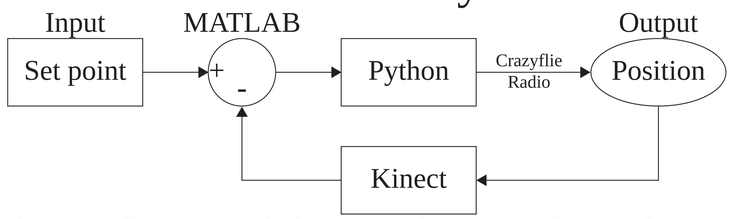
Automatically Controlling a Quadcopter

By A. F. Bower & J. Song

Goal

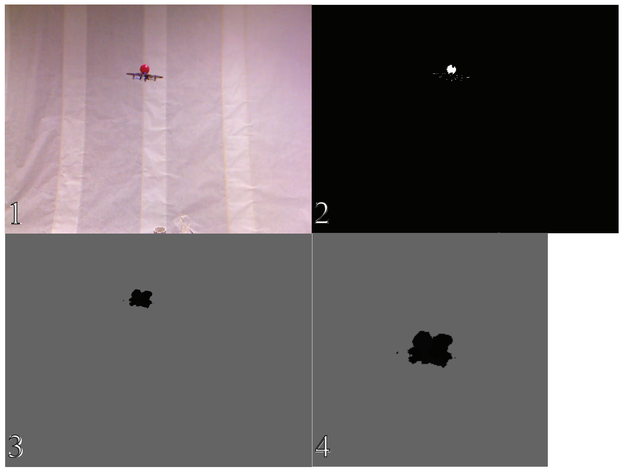
Quadcopters have very high maneuverability and agility. However, it is difficult to fly very small quadcopters, as even a small change in controls or air currents can greatly impact their behavior. The goal of this project was to allow a computer to control the quadcopter, which would allow for easily reproducible results.

Control Theory



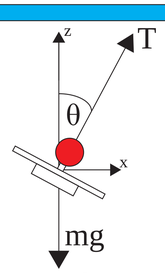
Before the controller is started, the Kinect determines the initial starting position of the quadcopter. Then, the user inputs the set point, which is the desired location for the quadcopter. This data is passed into the MATLAB function, which calculates the necessary thrust, yaw, pitch and roll given the difference between the current location and the set point. The MATLAB function is called by the Python code, which takes the output of the function (the control parameters) and passes it to the quadcopter. The quadcopter will change its position, at which point the Kinect will take another picture, updating the position. The position is passed through a low-pass filter to reduce noise; the filtered data is used by MATLAB. The process then starts over again.

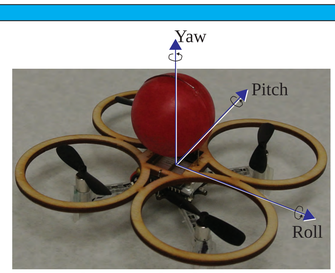
Image Processing



The controller calls for two pictures from the Kinect: a color image (1) and a depth image (3). First, the color image is converted into a black and white image in order to distinguish the red table tennis ball from everything else, where white pixels signify red (2). The center of the white pixels is approximated, which gives the approximate location of the quadcopter in two dimensions. To get the third dimension, the depth image is used. Since we know the approximate location of the quadcopter, we can crop the depth image to reduce computation time (4). This gives the location of the quadcopter in all three dimensions. Note that the shade of the depth pictures have been altered to increase visibility.

Theory





In two dimensions, given the free body diagram above, the behavior is as follows:

$$mx'' = T*sin\theta,\ mz'' = T*cos\theta - mg \rightarrow mx'' = T*\theta,\ mz'' = T - mg$$

x'' denotes the second derivative of x. The linearization applies for small values of θ.

The PID controller makes the following true; note that for the sake of simplicity, we are ignoring the integral term:

$$\theta = -k_{p,x}(x-x^*) - k_{d,x}x',\ T = T_{liftoff} - k_{p,z}(z-z^*) - k_{d,z}z'$$

The k's are coefficients, the asterisk denotes the set point, x' means the first derivative of x, and $T_{liftoff} = mg$ is the thrust necessary to stay level when θ=0. Combining the equations above, we find that:

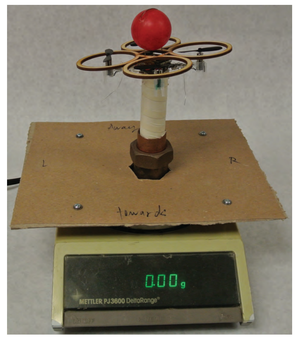
$$x'' + k_{d,x}gx' + k_{p,x}g(x-x^*) = 0$$
$$z'' + k_{d,z}z'/m + k_{p,z}(z-z^*) = 0$$

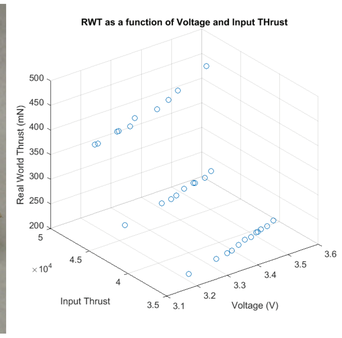
These second order differential equations closely resemble the behavior of a damped spring-mass system. Note that T ≈ mg; this approximation is made when examining x, but not for z.

In actuality, the PID controller controls the input thrust, *input*, not the real world thrust, *T*. Currently, the real world thrust is assumed to be linear with respect to input thrust and voltage of the battery, resulting in a planar surface:

$T(v,input) = a*v + b*input + c$, where a,b,and c are constants

Calibration





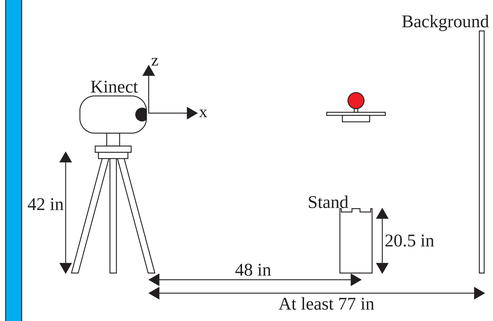
To calibrate thrust, the quadcopter was set up on a scale, as shown on the left. The lift generated was measured for various input thrusts and battery voltages. The results are shown on the right: the plane that best fit the data had the equation:

$$T = 202*vbat + 0.0132*input - 913.88$$

T is in mN, *vbat* is in volts, and *input* is between 0 and 60000.

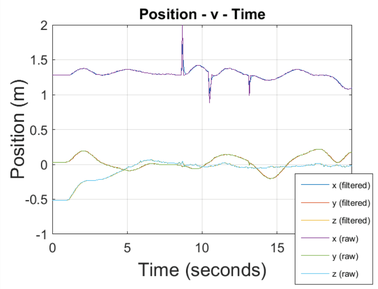
The k's in the PID controller were found through trial and error. k_p's were found first: they need to be large enough to avoid substantial drift, while small enough to not cause the quadcopter to fly out of the frame. k_d's were found next; they were picked primarily by how fast oscillations died out. k_i's were set last, and they were chosen based on how well the quadcopter remained near the set point.

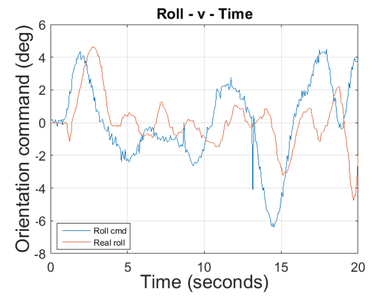
Setup

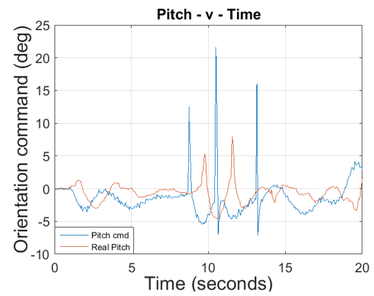


The setup is shown to the left. The Kinect must connect to an adapter that is powered by an outlet and connects to the computer. The computer must also have a Crazyflie radio in one of its USB ports. It is best to have the Kinect and the stand as level as possible.

Results







The quadcopter can maintain a single altitude very accurately, and the controller works well enough that the quadcopter remains within the field of view for at least a minute when hovering. When the set point moves in a pattern, the quadcopter is able to follow it loosely: the pattern is recognizable, though the quadcopter often overshoots or falls behind.

The controller has several shortcomings. There is a noticeable delay between the pitch and roll command sent to the quadcopter, and the actual pitch and roll. This delay causes the command and actual to have opposite signs later in the flight, leading to oscillations emerging. Furthermore, the controller is very fickle. Python seems to struggle to communicate with the quadcopter when it is not fully charged, and the Kinect occasionally loses the quadcopter, when it very clearly should not, as seen in the graphs.

Future / Continuing Goals

Currently, MATLAB only supports Kinect for Windows Version 1. Hopefully, they will update to support Kinect Version 2, which has a larger field of view, higher resolution and improved depth sensor, which will all substantially increase the space the quadcopter can fly in, as well as the reliability of the sensor.

The mathematical model should also be improved. Simulations based on the current model are too simplistic and idealized; a more realistic model will allow for more accurate simulations, which will greatly aide in optimizing the controller.