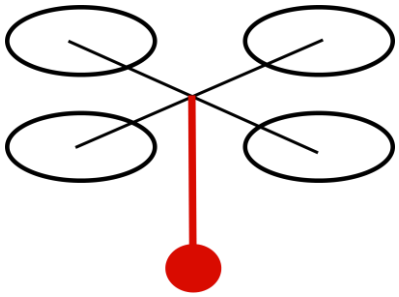


The Foliage Extracting Tele-Controlled Helicopter (F.E.T.C.H.): Computer Science Report

[4/30/2013]

Authors:
Cable Johnson
Theora Rice

University of Idaho
Moscow, ID 83844-1010

Table of Contents

Executive Summary

The Foliage Extracting Tele-Controlled Helicopter (FETCH) is a quadcopter that has been designed for the purpose of collecting “sun-foliage” branch samples from treetops. The needs of this project include that it be able to fly above the test trees and be reliable enough to collect samples more efficiently than current methods. With this in mind, Team FETCH has designed a quadcopter capable of flying over trees, with an extendable arm cutting attachment. At the bottom of this arm are cut-and-hold sheers, which are capable of separating and keeping branch samples until they can be dropped at a convenient location by the operator. The benefits of this solution include that it is more economic and efficient. The bulk of the cost of the FETCH system lays in its construction, with minimal maintenance funding required throughout its use. Efficiency is maximized by this system as well, in that it utilizes integrated cameras so that the operator can quickly find and collect a prime sample. The technical merits of this project exist in the modularity of the system. Both software and hardware within FETCH have been kept as simple and independent as possible in order to allow for easy expansion and repair. Business merits include that it is cost-efficient, and creates a prototype for further development.

Background

The motivation for the FETCH project resides within the University of Idaho College of Natural Resources department. Dr. Katy Kavanagh and her students conduct climate change studies on "sun-foliage" branch samples. "Sun-foliage" is the treetop flora that has received the most sun exposure, and, during pre-dawn hours, has the best water retention rate. Together, these aspects reflect the best current data about overall tree health. Currently, these samples are collected either by using a canopy crane, hiring tree-climbers, or using shotgun shells in an attempt to blow the branches off. None of these solutions is efficient, and most are expensive. Thus, the need exists to develop a safe, effective, and inexpensive solution. Team FETCH was assigned to solve the problem using some type of helicopter-themed design that could be remotely operated to collect branch samples. The expected benefits of this solution are that it will be more time and cost efficient.

At the beginning of this project, six students were assigned. Two came from electrical engineering, two from computer engineering, and two from computer science. The two students in computer science are Cable Johnson and Theora Rice. Prior to this project, their experience resided mainly in pure software programming, network architecture, and computer security. Throughout the course of this project, however, their computing horizons were broadened greatly. The following report sections detail their project experiences, the problems they overcame, and their future recommendations.

Problem Definition

The goal of the FETCH project is to provide Dr. Kavanagh with a working prototype of this quadcopter-based system by the end of the fall 2013 semester. This prototype will include the working quadcopter flight frame with extendable cutting attachment, a ground station laptop and radio controller, documentation on every aspect of the project, and the training necessary for operation. These deliverables will be produced only after extensive flight testing to ensure safety and effectiveness.

Concerning the software aspect of the FETCH project, the computer scientists of the team aspired to make the programming code small, modular, and efficient. This is primarily because small, pragmatic code is the least likely to have bugs, and therefore the best to use when directly interfacing with hardware. Keeping simplicity a priority also meant that the programmers' would be at less of a disadvantage in entering the world of microcontrollers, with its new libraries and various macros. Due to the short nature of the computer science senior capstone program, it was decided early in the semester that a pre-programmed flight board would be ordered in order to save time and effort. The PIC32 microcontroller in charge of LEDs, arm extension motor, and cutting motor, however, was specific enough to the project that programming was needed. Thus, Cable and Theora set a goal of finishing the code necessary for button LED control and radio-controlled arm extension by the end of spring 2013, as well as provide the code basis for cutting control.

Project Plan

The overall goal for the software portion of this project was to program the microcontroller that would control the extending arm and cutters, as well as the LED's. These three components would be controlled as follows:

1. LED's would respond to a push-on/push-off button on the quadcopter itself
2. The extending arm would be controlled by a DC motor rotating a worm gear
3. The cutters would be similarly controlled by a DC motor

Programming the microcontroller entailed providing full control of these three components. The planning for the LED's on/off button was somewhat trivial, but controlling the DC motors was where the challenge came in. Since quadcopter transmitters produce PWM signals, the microcontroller would have to read those PWM signals through the receiver and use them to decide which motor to spin in which direction.

Concepts Considered

As far as the LED's were concerned, there were two options for control. One was to be able to switch the lights on and off through the transmitter. This would allow for the state of the lights to be in the user's control whether the quadcopter was on the ground or in the air, providing more versatile functionality. The second option was to have the LED button on the quadcopter itself, so it could only be switched while the system was landed on the ground. This would provide less control over the lights, but would be easier to implement.

With regards to the DC motor control, there were also two options to consider. The first would utilize the fact that the transmitter outputted PWM signals, and DC motors take PWM signals as input. The initial thought was to feed the signal from the transmitter directly into the DC motor, thereby undermining the need for a microcontroller at all. However, there was a concern that after doing some preliminary tests that the signal coming from the transmitter made the motors spin a bit faster than desired. This could, in turn, burn out the motors entirely over time. So the other thought was to send the PWM signal through the transmitter to be preprocessed, and reduce the frequency before feeding it to the DC motor, to ensure full control over the motors themselves.

However this brought up another question: could the PWM signal simply be sent through a translation function to reduce the frequency, while retaining full control over the motors? If the microcontroller were to be used, the two options for controlling the motors come down to translation and interpretation: the translation method, sending the PWM signal through a frequency reduction and then feeding it straight to the DC motor, and the interpretation method, simply writing a signal with constant frequency depending on the frequency of the signal being received.

Concept Selection

Regarding the LED control, there was the option of either having the switch on the transmitter, or on the ground. However, in the end it was decided to put the button on the quadcopter itself so that the LED can only be switched on and off while the system is landed. This is to prevent the user from accidentally turning off the lights while the quad is in flight, thereby losing all visuals in the predawn hours. This method will ensure that the state of the LED's will not change while the quad is in flight. When flying in the darkness, the user will turn on the LED's before takeoff, and when flying in the daylight, the user can choose to leave them off for longer battery life.

Though it would be possible to directly control the DC motors with the transmitter, this would cause the motors to spin too fast and in the long run potentially burn them out. Thus, the method of choice is to feed the PWM signal into the microcontroller for preprocessing before spinning the motors. This ensures that the motors are rotating at a speed that is not detrimental to them over time, resulting in greater longevity. This also allows for the writing of a motor speed that makes it easiest to control the arm.

Once that was decided, the question came to whether translation or interpretation was the better choice. Though translation seemed to be easier, since it would only entail sending the signal through a frequency reduction function, there were some problems with this method in preliminary testing. Namely, the signal coming from the transmitter was not a constant one. Though the frequency fell into a general range, it was still often sporadic and changed without warning. This would result in erratic movement if the signal were sent directly to the motor. So the chosen solution was to go with the interpretation method, which entails three steps:

1. Read the PWM signal
2. Interpret the PWM signal, aka determine the *range* of frequency rather than the frequency itself.
3. Write a predefined constant frequency to the motor

Essentially, in the code, if a high frequency is received (a frequency above a predefined threshold) the motor is rotated one way, at a steady speed. If a low frequency is received (frequency below another predefined threshold) the motor is rotated the other direction. If a mid-range frequency is received (in between the high and low bounds) the motor is set to halt. So despite the fact that inconsistent frequencies are being received, a smooth, constant frequency will always be written to the motors.

System Architecture

Once the software design was finalized, Cable and Theora began with the beginner-level task of setting up the LED button circuit. To do this, they utilized an outside 5 volt power source, and three of the PIC32's pins: one input, one output, and one ground. These were located on a convenient port that was removed from other, more specialized pins that may be needed in later development. The button attachment is set up between ground and the PIC32, in order to provide user input (For a more detailed view of the Button/LED circuit, please see Appendix A.) Once the PIC32 has sensed that the button is been pressed it will invert the output pin, which is attached to a white, high-intensity LED. The inversion will cause the light to invert its state (either on or off), and maintain that state until the button is pushed again. This design requires a minimum of PIC32 resources, and can share the same power supply that the extension motor requires. Thus, it is an effective, efficient solution.

Detecting PWM signals and using them to indicate arm extension was segmented into sequential tasks. The first was to read the signal from an attached receiver, the second to interpret it, and the third to write a new signal to the extension motor. Until a PWM signal is sent and read by the PIC32, state constants remain unset and the interpretation and writing steps cannot execute. To facilitate this design, two ground pins, one input pin, five output pins, and a built-in PIC32 timer are utilized.

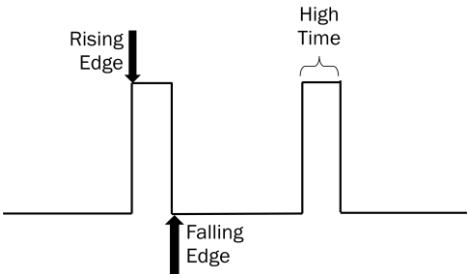


Figure 1

Reading consists of sensing a PWM signal being sent from the receiver to the input pin of the PIC32. (For a detailed circuit diagram of the receiver, please refer to Appendix B.) When a signal is detected, the function ReadPWM(*input_pin*) captures the running timer value at the rising and falling edge of the signal, as depicted in figure 1. From these timer values an overall "high time" is calculated, which reflects the frequency of the signal. This frequency is used in interpretation in order to decode the needed motor direction.

In InterpretPWM(), the frequency of the PWM signal is measured against defined bounds. If the high time is above the high frequency threshold, a motor state variable is set to 1. If it is below a low frequency threshold, the variable is set to 2. These, in turn, correspond to different directions the motor may go. The motor itself is connected to the PC32 via an H-bridge. This bridge uses six pins to run the motor, and one of these is called the direction pin. When the motor state variable is set, that pin is pushed either high or low, depending on the variable number.

Once the signal has been sensed and direction determined, WritePWM() will write a steady PWM signal to the H-bridge, which will cause the motor to turn the desired direction at a steady rate. (For a detailed circuit diagram of the motor output, please refer to Appendix C.) It is within this function, primarily, that bounds are checked against a programmed counter. Minimum and maximum bounds are defined and compared against in order to prevent over-extension or under-retraction of the arm hardware. This is a safety feature, in order to prevent damaging the motor or arm framework. If the counter is within safe boundaries, the motor is ordered to spin, and the counter is either incremented or decremented. The spinning will continue until either the user has ceased sending signals, or the counter has reached the minimum/maximum value.

By breaking the PWM signal actions into three separate steps, Cable and Theora have ensured that the code is modular and easy to understand, as well as portable. Though the mechanism for controlling the cutting action of the sheers has not yet been determined, ReadPWM(*input_signal*) and InterpretPWM() can both be used. Their purpose is simply to detect, read, and interpret the frequency of the signal sent from the radio controller. If the chosen mechanism is another DC motor setup, then WritePWM() will need minor changes to which output pin the PWM signal is sent to. In the case that a pneumatic system is chosen for the final design, then a new write function will need to be integrated into the programming base.

Major Problems Overcome

Upon being assigned to the FETCH project, Cable and Theora expected to have a difficult challenge in learning how to directly program hardware. However, this initial expectation was compacted by a lack of funding until mid-semester. When the microcontroller was finally purchased, they immediately began to implement and test their software design.

The first problem arose in the form of what programming IDE to use. **`mpide`** was recommended for use with the MX3CK PIC32 that had been purchased, as it did not require an external programmer. After two weeks of experimentation, however, the computer scientists were not able to successfully utilize the software to program the PIC32 to affect external sources. On the suggestion of their team mates, they switched to using the **`MPLab`** IDE with a borrowed programmer. By choosing to switch to this new IDE, they were able to get much more assistance from their fellow team mates to expedite the learning process.

Another hurdle existed in the new peripheral library "plib.h" that is employed in the development of microcontroller programs. Unlike other C libraries which have well-known

resources, it took Cable and Theora a few days to find an adequate description of what the library contained. Once this documentation was found, the macros still contained some obscure variables and references. This slowed the process of programming down exponentially, as Cable and Theora often had to track down one of their other team mates in order to discern the meaning of a library function.

Reading PWM signals also provided a large obstacle for the programmers, as it was a action that their team mates had never done before. After being given a crash course in what a PWM signal is, they attempted to utilize a `captureinput` macro, which did not end up reporting the correct values. This is because it would detect more than the first instant of a rising or falling edge. It took daily meetings over the course of two weeks in order for Cable and Theora to work through the logic of how to write their own functions. Once this was done, they had to conduct tests in order to set the correct frequency bounds for the FETCH product.

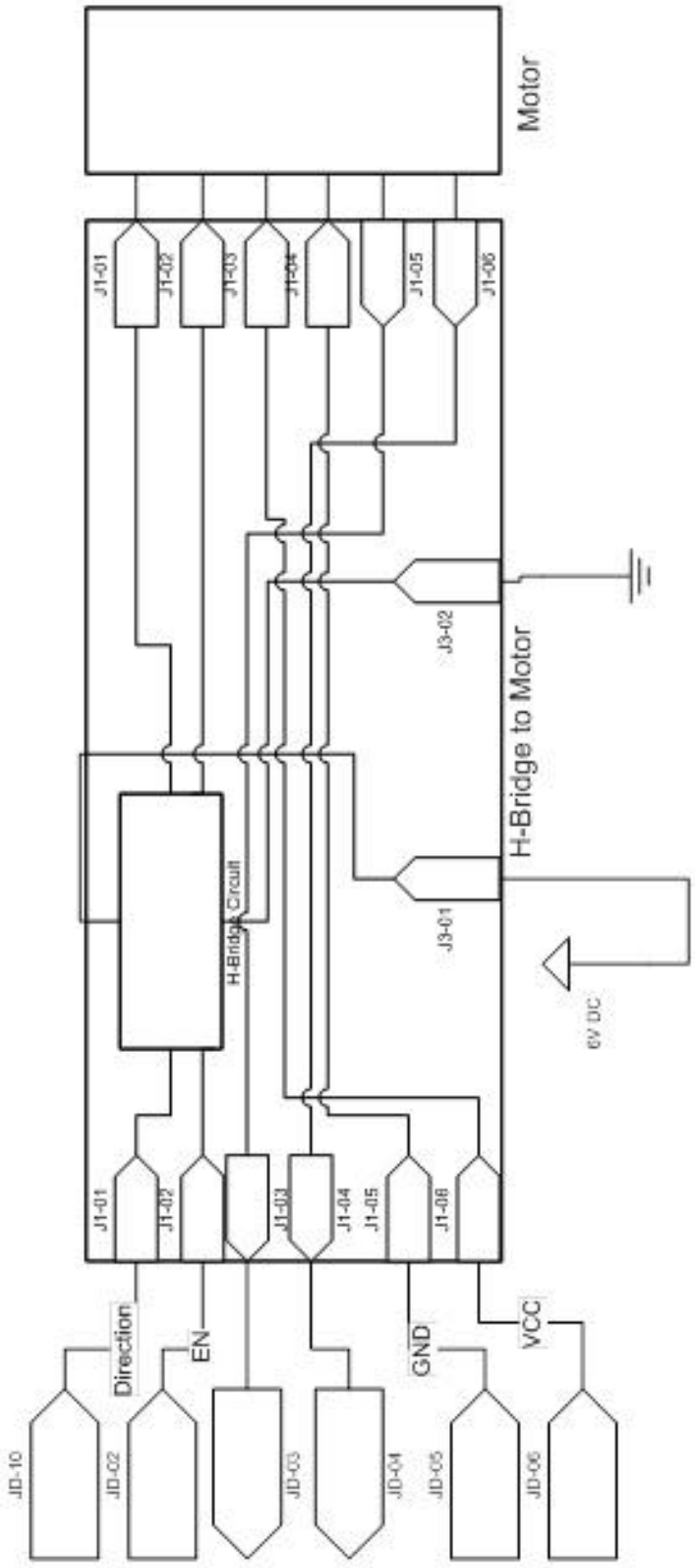
The overall greatest difficulty the computer scientists experienced was learning how to use and troubleshoot hardware. By beginning with the LED/Button setup, they were given a basic introduction to how circuitry works. When they began to work with PWM signals, they had to quickly learn how to use a separate power supply, function generator, and oscilloscope so that they could ensure the hardware would properly reflect their software. This lack of knowledge about hardware and circuitry slowed the project down immensely, and in one case jeopardized the hardware system. This was when, in an attempt to optimize system power, Cable and Theora increased the voltage generated by the power supply until one of the LED/Button circuit resistors combusted. When the resistor was replaced and the system tested again, nothing appeared to be working. However, with the quick assistance of Kora Barnes (Electrical Engineering student), they managed to narrow down the problem to a shorted pin on the PIC32 board. Once the program was changed to use a different input pin for the button, everything was once again working.

Future Work

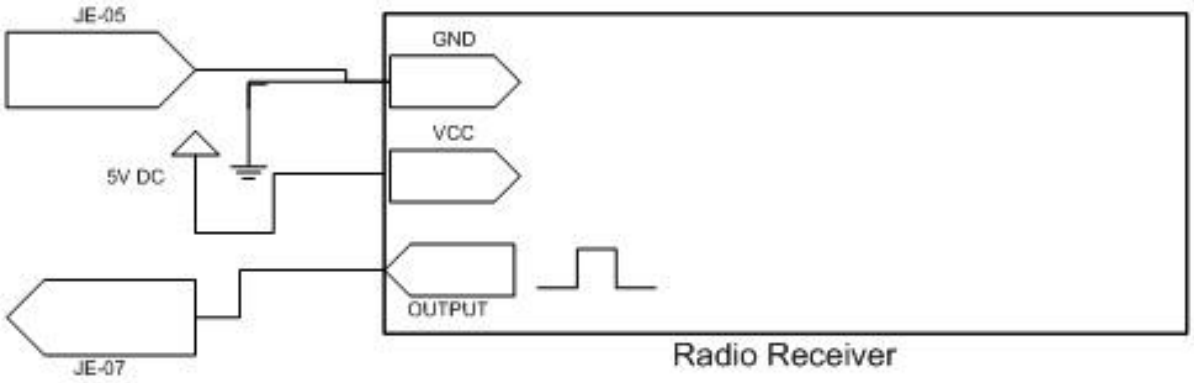
At the end of the spring 2013 semester, Cable and Theora have left their team mates with fully-operational software that can be implemented on the final prototype. What has been left for future development mostly has to do with the cutting mechanism. As of this date of writing, this mechanism has not been solidified, and cannot be fully programmed for. However, within the software, the computer scientists included “stubbed” code that can be easily modified and used for this future functionality.

Other software aspects that the project may need to include in the future would be the ability to construct a rotating cutting mechanism. This would be so that the shears could easily modify their angle to gather branch samples. For the first prototype of the system, however, this is unlikely to be implemented.

Appendix A


Motor
J1-01
J1-02
J1-03
J1-04
J1-05
J1-06
J3-02
H-Bridge to Motor
J3-01
H-Bridge Circuit
6V DC
J1-01
J1-02
J1-03
J1-04
J1-05
J1-06
Direction
EN
GND
VCC
JD-10
JD-02
JD-03
JD-04
JD-05
JD-06

Appendix B



Appendix C