WiiCopter

EECS 149/249A Final Report

Jimmy Su, Steven Campos, and Gangbaolede Li

ABSTRACT

This paper describes a project that aims to control a quadcopter in an innovative way. Many engineering principles are exemplified along the way, and they demonstrate that modelling, design, and analysis are interlinked concepts in well-orchestrated projects.

1 DESCRIPTION

The goal of this project is to implement the Nintendo WiiMote as an input device in dictating the motion of a quadcopter. Additionally, to account for the prevalent safety issue of the quadcopter flying into an obstacle, this project will implement a rudimentary safety system for obstacle avoidance; namely, the drone will detect walls in advance and slow down/stop in sufficient time to prevent collision, all through the use of proximity sensors.

2 IMPLEMENTATION

2.1 Data Flow

On the physical layer, the WiiMote is connected to the laptop over Bluetooth, while the laptop and mbed share a Wifi network originating from the drone. The purpose of the laptop is to extend the WiiMote's range since Wifi fairs better than Bluetooth over longer distances.

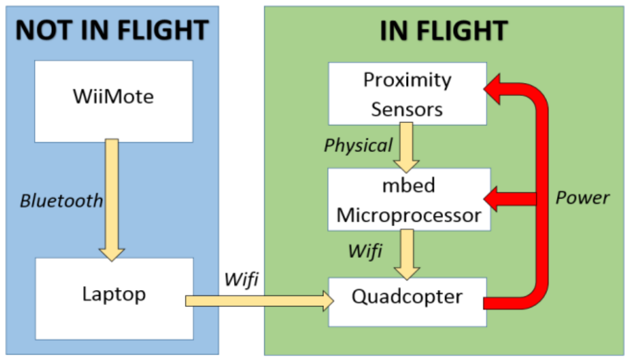


Figure 1: Block diagram of physical connections

User input data begins at the WiiMote with button presses and accelerometer fluctuations. This data moves from the Wiimote to the laptop over a L2CAP bluetooth connection, which is the WiiMote's only method of communication. The laptop then immediately forwards the WiiMote data to the mbed using the UDP protocol, which is connectionless.

The mbed is the main decision maker for the quadcopter, which is why all data flows to it. Upon receiving both WiiMote and sensor data, the mbed determines what the best action is and issues a command directly to the quadcopter using the UDP protocol.

2.2 Operation[1]

The operation of the quadcopter encompasses fairly standard motions, such as flying forward, reversing, and turning. For these movements, the laptop will calculate the roll and pitch of the accelerometer values output by the WiiMote. When those values are relayed to the mbed, it will translate them into appropriate actions for the drone to exercise. The subtlety here is that the roll of the WiiMote affects the yaw of the quadcopter, rather than its roll. The design reason behind this choice is so that the on-board camera is always facing forward, akin to driving a car; this allows the user to be less dependent on being aligned with the drone.

Aside from actual movement, other operations include the self-explanatory take-off and land functionality, as well as useful features such as calibration and emergency toggle. These operations are controlled via the buttons on the WiiMote, in contrast to the previously-mentioned flight motions that are controlled based on accelerometer values. A side feature is the ability to adjust the altitude at which the quadcopter flies, which are also controlled via buttons.

In regards to safety, there will be proximity sensors along the front and sides of the quadcopter to detect obstacles and move away from them.

2.3 Construction

Ideal construction of our WiiCopter includes mounting two sensors in the front and one sensor per remaining side. We feel this is the ideal amount and placement

[1] Video of our project in action: https://www.youtube.com/watch?v=FBJBY91HUqU

because no side should be blind, allowing the user to bump into a wall. Moreover, additional sensors beyond these five add precious weight. The front makes better use of two sensors since the user may have the WiiCopter flying in an arc instead of in a straight line.

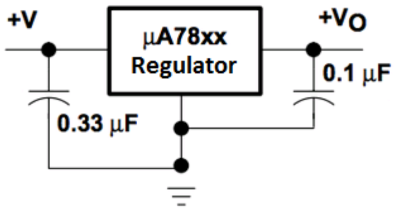
Once the sensors are mounted, an mbed and breadboard are similarly mounted to the WiiCopter's hull (being careful to keep the drone balanced). The drone battery is then connected to two jumpers, powering the regulator circuit on our breadboard (see Section 2.4).

2.4 Power

One design aspect of our project is the fact that the mbed needs to be mounted on the quadcopter, for two reasons:

1. The sensors are mounted.
2. The sensors are not wireless.

As a result, voltage will need to be supplied to three varieties of components: the KL25Z mbed microprocessor, the CC3000 Wifi module, and the proximity sensors. Regarding the specific voltage to supply to each component, the mbed and the Wifi module have a fairly broad range of acceptable voltage inputs – 4.5V - 9V [1] and 2.7V - 4.8V [2] respectively. However, the voltage range of the sensors is a restrictive 4.5V - 5.5V [3], and thus we decided to regulate a reliably higher voltage down to 5V. Initially, this voltage source was an external 9V battery, but we modified this design for weight reasons (more detail in Section 3.1). We instead took advantage of the on-board AR Drone battery, which has three 4V cells to supply a total of 12V to the drone; we utilized two of the cells to simultaneously power the regulator circuit with 8V. (It should be mentioned that we did not feed all 12V to the circuit due to power limitations of the regulator device itself.)



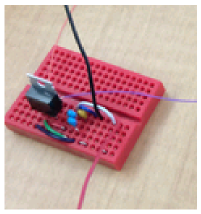


Figure 3: 5V regulator circuit

Figure 4: AR Parrot Drone 12V Battery

2.5 Logic

The logic of the quadcopter operation is best illustrated by the state machine in Figure 2. Once we calibrate in our initial state and subsequently take flight, we enter a composite state that has a reset transition within the state to HOVER and a preemptive transition out of the state to LAND. The reset transition is appropriate so that the quadcopter does not immediately fly off upon take off, and the preemptive transition prioritizes the LAND state – preventing the lag between when the user desires the quadcopter to land and when it actually lands. The latter aspect is especially useful because it allows the user to immediately prevent the quadcopter from succumbing to any possible dangers.

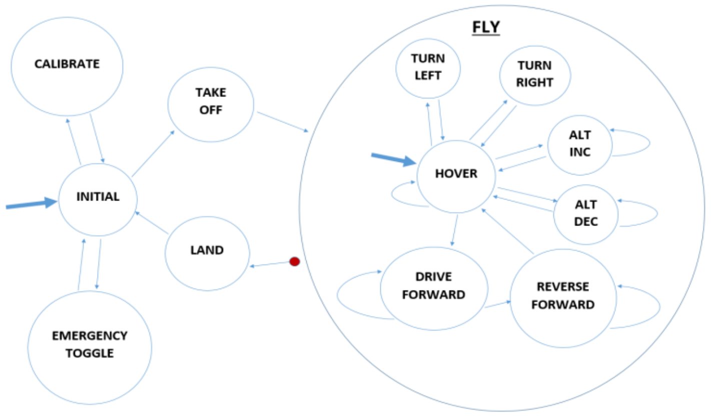


Figure 5: Composition State Machine, with preemptive and reset transitions

The astute reader will notice that turning and driving are separate states. We elected to pursue this arrangement because an evasive maneuver that is appropriate for drive (just reversing) may not be appropriate for turning.

3 LIMITATIONS AND THEIR SOLUTIONS

3.1 Weight

The most fundamental limitation of our project is the extra weight that the quadcopter can carry: if our mounted circuitry is too heavy – no matter how robust it is, the project will fail or at the very least drain the battery and wear out motors more quickly. Keeping this in mind, we conducted some weight tests of our own, in which we added some dummy weight and gradually incremented it until the quadcopter failed to be practical. This payload bound of ours was about 120 grams. Note that this may differ from a higher, theoretical out-of-box limit, due to previous usage of the drone by other groups in previous years [4].

During our implementation, we trimmed weight where we could: we sliced breadboards, shortened screws, and utilized wire wrap to reduce solder. In fact, weight was a deciding factor of why we chose to use the mbed processor – which we weighed to be 21 grams – as opposed to other processors, such as the Raspberry Pi – which can weigh as much as 45 grams [5].

However, our optimizations could only lower us to 138 grams, which still exceeds our experimental threshold. We realized that minor tweaks will not push us below our limit, so we made a considerable change to utilize the already-present AR Drone battery in place of the secondary 9V external battery, as mentioned in Section 2.4.. This provided a 46 gram reduction in weight, which fortunately dropped the total payload to an acceptable 92 grams.

3.2 Sensors

The IR proximity sensors used on the quadcopter presented challenges beyond just their weight. The first of these problems stems from the fact that these sensors are active rather than passive, meaning that they emit and affect the world around them instead of just passively listening. During our project, we noticed something that we coined “sensor coupling”: where two sensors could each assist the other’s reading.

For example, take the left front sensor: normally, by itself, it would read the light emitted from its IR led that bounced off of an object and into it’s receiver. When placed near another sensor in a similar orientation, the left sensor now bases its readings off of the light emitted/reflected from **both** IR leds. The solution to this problem was to slightly angle the sensors away from each other, as well as experimentally determining what new distance-to-voltage curve the sensors operated on when interfering with each other.

Another challenge we faced was that the sensors had a non-uniform sensitivity. We noticed that at very close distances, the sensor voltage fluctuated more wildly for a small change in distance. Therefore, to ensure adequate precision, we chose sensors with higher sensitivity over the range we wanted to measure. Since we wanted to detect objects about 30 cm away and further, we chose sensors with an appropriate range for that. Moreover, while the sensors do begin to lose sensitivity when objects were closer than 30 cm, they still address an important qualitative issue: detecting when something is too close, even without knowing its exact distance.

3.3 Latency

Latency was a serious consideration for our project, given that we need to respond to a user and address the issue of safety. On a technical level, latency is the time it can take for data to flow from one point to another, but in more practical terms we are asking ourselves: “How responsive is this?” A quadcopter with a 2-second reaction time would have a hard time avoiding walls, unless it moves at an extremely slow speeds – in which case it would be quite boring to control.

We first addressed latency in the way we moved data around, which, as mentioned, in Section 2.1, is primarily UDP. By using UDP to send data across the Wifi network, we eliminate the need for devices to establish a connection with each other and develop a more real-time sense of communication. While TCP ensures that data arrives – and in order, it involves much repetition of data and handshaking to facilitate those benefits. Data not arriving was not an issue with UDP because of the level of repetition we use for appropriate actions. For example, if the user is depressing the take-off button, it’s perfectly acceptable to continuously send takeoff packets until the button is released. To address the issue of data arriving out of order, we label our data with a sequence number which should always be increasing.

Another key area to reduce latency was our device/programming choice. By using an mbed, we gain access to a more real-time device. The mbed can live up to guarantees in terms of execution time, whereas alternatives such as a laptop or Raspberry Pi cannot since they run on operating systems that schedule tasks as they see fit. However, since even an mbed is subject to hiccups in terms of timing (a task like reading/sending data may take longer than expected), we keep track of the time of execution in our code’s control loop. If we notice that our frame rate (how often we make and issue a decision) is dropping, then the mbed can take appropriate action such as landing the copter or issuing the user a warning.

Finally, after mitigating latency everywhere possible, we account for any remaining latency issues by making the mbed the primary decision maker for the quadcopter. This way, if data is not received from either the WiiMote or the laptop, the quadcopter is not left without a valid action to take.

4 TESTING

The course staff have always made the distinction between someone who is an engineer, and someone who just plugs in their creation to see if it works before returning to the drawing board. Keeping that in mind, we chose to be engineers and gather data in a controlled manner before proceeding. This section will outline our exploration of our hardware, its abilities, and our attempts to intelligently test it.

4.1 Latency

In order to get a better idea of what latency we could expect from the components before actually allowing the quadcopter to fly, we tried to measure responsiveness at every key point in the setup. Our approach was to send data on a round trip between two devices, measure the elapsed time from departure to return, and divide that in half. Repeating this many times (automated), we could compose ideas of each connection's average and worst-case latency. Summing these latencies like resistors in series gave us a fairly realistic sense of the system's reaction time.

4.2 Quadcopter Abilities

Knowing the quadcopter abilities is important for determining the distances we use for safety. For example, initiating obstacle avoidance at a distance of 50 cm does us no good if the quadcopter takes 60 cm to change direction. For that and various other reasons, we ran the quadcopter through exercises where we tested its abilities on video while carrying weight. This gave us an opportunity to measure the ability of the quadcopter to stop and change direction, leading us to believe 30 cm was a safe enough distance at which to trigger our sensors.

4.3 Sensors

Testing of sensors was simple, but it was also necessary in order to verify what the manufacturer says. After all, electronic components can vary from batch to batch, and our environment could have an unanticipated effect on the sensors. After running multiple static tests on the sensors with a multimeter, we saw that (for most ranges) the sensor output accurately matched the datasheet. Though, one interesting discovery was that different surfaces could affect our readings.

4.4 Debugging/Simulation

To facilitate debugging, we used the laptop as a simulation tool. A stand-in, for any device on the network. By running the appropriate program to simulate a device, we could have the laptop send/receive data in a more predictable manner, and make observations about the data it comes across.

5 ACKNOWLEDGEMENTS

Enormous thanks to Antonio Iannopollo (our main GSI) for being with us every step of the way, as well as Ben Zhang and John Finn for answering any other questions we had.

We are additionally very grateful to professors Edward Lee and Alberto L. Sangiovanni-Vincentelli for giving us deeper insights into the world of embedded systems, as well as solidifying our understanding of the relationship between modelling, design, and analysis.

6 REFERENCES

[1] http://developer.mbed.org/handbook/mbed-FRDM-KL25Z

[2] http://www.ti.com/lit/ds/symlink/cc3000.pdf

[3] https://www.sparkfun.com/datasheets/Sensors/Infrared/gp2y0a02yk_e.pdf

[4] http://forum.parrot.com/ardrone/en/viewtopic.php?id=260

[5] http://en.wikipedia.org/wiki/Raspberry_Pi