OBJECTIVES

In this lab you will DO the following:

- Attach the gondola (PCB and fans) to the blimp envelope
- Control the motors
- Control the IR proximity sensing subsystem
- Develop a PID controller
- Implement autonomous altitude control

In this lab you will LEARN the following:

- How to work with the multichannel timer peripherals on the F5510
- How to control motors using PWM and an H-bridge driver IC
- Theory of IR proximity sensing/ranging

DELIVERABLES

At the beginning of the next lab period you will demo your wireless controller, and describe/present the organizational structure of your firmware. This demo will not be graded; the purpose of the demo will be to check in with the TA and keep your project on track.

RESOURCES

These documents and web resources will be useful in completion of the lab and/or in answering the questions posed. Additional resources are described in the procedures below.

- BlimpBot schematics and component datasheets on course website
- Task control block example from lecture
- MSP430F5510 Product Page
IMPORTANT NOTES

There are several very important rules to follow when using the blimp hardware. If you do not follow these rules, there’s a good chance you’ll release the magic smoke and have to wait to have your hardware repaired.

1. **ALWAYS POWER ON THE BLIMP BATTERY SWITCH BEFORE CONNECTING THE DEBUGGER.** Never connect the debugger when the battery switch is OFF or when the battery is disconnected or dead.
2. **NEVER short the motor terminals.** The motor driver ICs and battery are very good at sourcing HUGE currents, and things will be damaged.
3. **ALWAYS plug in the battery with the proper polarity.** There is no protection against a reversed battery!
4. **ALWAYS keep the battery voltage above 3.4V.** When you notice the regulated voltage on VDD starting to drop below 3.3 V, it is time to charge the battery (Perhaps your firmware should watch for this “drop out” condition...)
5. **NEVER attempt to do anything to your LiPO battery without talking to the TA.** LiPOs are extremely dangerous. If your battery appears to be having problems, email the TA. Do not try to measure or modify your battery or this could happen.

PART 5: ATTACH GONDOLA TO ENVELOPE

Now that the framework for remote control of the blimp has been developed, you are ready to attach the gondola to the envelope, balance the blimp, and attach ballast to achieve a neutrally buoyant blimp.

**Figure 1.** Attach the gondola to the envelope as shown using monofilament and tape.

**SUGGESTED METHOD FOR ATTACHING GONDOLA:**

1. Get two 2’ lengths of monofilament (fishing line).
2. Pass each monofilament length through the two outer holes on the gondola such that the ends of the monofilament emerge on the top side of the PCB (side with battery).
3. Using **masking tape** (easy to remove), attach the gondola to the blimp envelope as shown in the figure above. The gondola should hang about 2” below the envelope.
4. Ballast the blimp by attaching paperclips, etc. uniformly to the front and back until it is neutrally buoyant.
5. Reposition the gondola until the blimp is balanced front to back. This ensures that horizontal motion will not cause the blimp to ascend/descend.
6. Once the blimp is balanced front to back and is neutrally buoyant, mark the monofilament and envelope at the four points where they meet.
7. Replace the masking tape with strips of packaging tape for a more permanent attachment. Align the marks on the monofilament and envelope. Be sure that the monofilament doubles back under the tape strips for extra security as shown in the figure.
8. Clip the excess monofilament. You are now done.
PART 6: MOTOR CONTROL

In this section you will develop a module for PWM control of the fan motors.

HARDWARE PRIMER

The three fan motors, titled LFAN, RFAN, and VFAN (Left, Right, and Vertical) on the schematic diagram, are each connected to one channel of an LV8405 H-bridge driver IC. The purpose of this IC is to provide high-current switching for motor control as well as to protect the rest of the circuit from the large voltage spikes produced by the inductive load of the motors. Inputs and outputs 1 & 2 of the motor driver make up the first channel of motor control, and 3 & 4 make up the second channel. The truth table below details the four modes of operation of each motor driver channel: Standby, Reverse, Forward, and Brake.

<table>
<thead>
<tr>
<th>IN1 (IN3)</th>
<th>IN2 (IN4)</th>
<th>OUT1 (OUT3)</th>
<th>OUT2 (OUT4)</th>
<th>Charge pump</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>H</td>
<td>Z</td>
<td>Z</td>
<td>ON</td>
<td>Standby</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td></td>
<td>Reverse</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td></td>
<td>Forward</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td></td>
<td>Brake</td>
</tr>
</tbody>
</table>

Figure 2. Truth table for LV8405 Motor Driver

Each channel of the motor driver IC is connected to a pair of pins on PORT1 of the MSP430F5510. Figure 3 shows the relation of the MSP outputs to each of the fan motors. In this diagram, “_H” and “_L” stand for the “High” and “Low” sides of the motor and correspond to the 1st and 2nd inputs for each channel of the motor driver. For instance, setting VFAN_H = 0 and VFAN_L = 1 will put the vertical fan in Forward. To initialize the motor control subsystem, you should drive all 6 output pins HI, which puts each motor channel in Standby.

Figure 3. Motor driver connectivity detail from main blimp schematic.

PWM DETAILS

Pulse-width modulation will be applied here to vary the speed of the motors. To implement PWM, you will need to configure each channel of the TIMERA0 peripheral, which is capable of producing four independent PWM signals with unique duty cycles. The goal is to configure the PWM in such a way that the motor driver quickly toggles between the Standby and Forward states (to go forward) or between the Standby and Reverse states (to go backwards). Your motor controller should NEVER switch to the “Brake” state, as it will cause unnecessary current consumption. Keep in mind that the period (1/frequency) of the PWM signal for every channel of TIMERA0 is controlled by the first timer module register (TA0CCRO). The PWM frequency for this application should be set fairly low (several hundred Hz to several kHz) as the motor driver may not respond well at high frequencies.
The vertical fan will not be possible to control in the same manner, as outputs P1.0 and P1.1 cannot produce PWM signals via direct hardware control. You will need to develop a creative solution to vary the speed (or the effective thrust) of the vertical fan motor. Periodically pulsing the motor or implementing a form of software PWM are two possible methods of varying the effective thrust. Keep in mind that software PWM can burn a lot of CPU cycles if the frequency is set high.

GOALS:

1. Implement a motor control task which can
   a. Update the direction and speed for horizontal fans LFAN and RFAN.
   b. Update the direction and effective thrust for the vertical fan VFAN.
2. Initialize the motors to a safe (off) state by driving all lines high during system initialization.
3. Allow control of the motors from your client application via the Set{} command outlined in the Phase 1 prompt

HINTS:

- If possible, don’t allow the motor driver to get into its “Brake” state. This state shorts the motor winding terminals in order to slow the motor down quickly, and will cause unnecessary extra current consumption. Electronic “Braking” is intended for use in friction drive applications like an RC car or a conveyor belt, and is not necessary here.

PART 7: IR PROXIMITY SENSING

In this section you will develop a module to control the IR proximity sensing subsystem. This sensor is comprised of five directional channels which allow for ranging of objects below, in front of, behind, and to the left and right of the blimp.

THEORY OF OPERATION

Similar to your E-field sensor from lab 3, this device has multiple transmitters and a single receiver. The transmitters are narrow-angle IR emitters (LEDs), of which four are mounted on the periphery and one in the center of the PCB. The receiver is a single wide-angle infrared photodiode mounted near the center of the PCB. The analog signal from the detector is buffered by an operational amplifier circuit and routed to P6.0 of the F5510 (Channel A0 of the ADC, labeled IR_RCV1_LO in the blimp schematic).

The intensity of the reflected infrared signal, as measured using the photodiode, will be used to estimate the range of a nearby object. As with the E-field sensor, the output will be toggled and the result accumulated via synchronous detection. However, there is one major difference between the E-field sensor and the IR proximity sensor in that the received IR signal received is NOT high-pass filtered; if we transmit a square wave, we will receive a square wave rather than the brief transient spikes produced by the E-field receiver. Also, the upper corner frequency of the IR detector is much lower than that of the e-field detector, so you will need to pause after toggling the IR LED before you sample the detector. An efficient way to implement this is to sample just before toggling the IR LED state (in contrast to the E-field sensor in which you sampled just after toggling the transmitter state).

It should be noted that reflectance of surfaces in the infrared spectrum can vary significantly, causing inconsistent ranging results with different surfaces.

CONNECTIVITY

Because the IR emitters need high currents (50-80mA) to produce adequate amounts of IR radiation, external drive circuitry is required. Each IR emitter has a dedicated driver circuit utilizing a Darlington pair bipolar transistor and a current limiting resistor.

Due to a lack of pins on this MSP430F5510, the IR emitters are controlled individually through a demultiplexer/decoder IC, the 74HC238. The signal to be transmitted should be produced by P2.0 on the F5510, which is connected to the Enable input of the 74HC238. A specific IR emitter may then be selected by configuring PJ.1, PJ.0, and P1.6 as specified in Table 1. Please reference the main blimp schematic for a better understanding of the IR driver and receiver hardware.
### Table 1. IR LEDs with decoder select values and emitter direction

<table>
<thead>
<tr>
<th>LED Number</th>
<th>{PJ.1, PJ.0, P1.6}</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED1</td>
<td>{0,0,0}</td>
<td>Forward</td>
</tr>
<tr>
<td>LED2</td>
<td>{0,0,1}</td>
<td>Reverse</td>
</tr>
<tr>
<td>LED3</td>
<td>{0,1,0}</td>
<td>Right</td>
</tr>
<tr>
<td>LED4</td>
<td>{0,1,1}</td>
<td>Left</td>
</tr>
<tr>
<td>LED5</td>
<td>{1,0,0}</td>
<td>Down</td>
</tr>
</tbody>
</table>

GOALS:

1. Fill in the “Sensing” task from Phase 1 by implementing IR proximity sensor polling.
2. The reception of a “Poll Sensor” message from the client application should initiate a single round of sensing. A general inter-task signal (for instance, from the motion control task) should also be capable of initiating a round of sensing.
3. The client side application should be able to identify which of the five sensor channels to poll (down, fwd, rev, port, and starboard) via the single byte payload of “Poll Sensor”. If another task initiates the sensor polling, it should also be able to specify which sensor channels should be read.
4. The message sent back to the client application can be up to ten bytes long, as specified in the Phase 1 prompt. It should include some indication of which sensor channels were polled and therefore which are valid in the returned packet.
5. The client application should be capable of displaying the received sensor data, either graphically or numerically.
6. **Optional:** If you find yourself frequently twiddling parameters of the sensor task (e.g., number of samples to average), make them remotely adjustable from the client application.

HINTS:

- The ADC10_A peripheral on the F5510 is NOT the same as the ADC10 peripheral you have used on the F2012. Register names have changed a little, and at least one configuration bit will be different.
- You can see the IR LEDs lighting up with your camera phone.
- Don’t leave the IR LEDs in the ON state between rounds of sensing… this will quickly drain the battery.

### OPTIONAL EXTRA CREDIT: LINEAR MAPPING OF RANGING DATA

As you will certainly realize after completing Part 7, the output of your IR proximity sensor is very nonlinear. In order to linearize the ranging data and make possible the use of more interesting control algorithms, you can map the output of your proximity sensor to a linear distance scale using empirical data from your sensor.

A look-up table (LUT) will be useful for this mapping purpose. You may choose to perform linear interpolation for sensor values that fall between entries in the LUT, or you may simply choose to round to the nearest LUT entry. Be sure to declare your LUT as “const”, or else it will be copied to RAM during initialization!

GOALS:

1. In your IR sensing module, add a LUT-based map (based on experimental data) which produces a linear distance reading given any possible sensor value.

HINTS:

- Be aware that different surfaces reflect/transmit IR differently. This has little to do with their visible color. Use consistent surfaces when collecting data for sensor linearization.
In this section you will develop a generic Proportional-Integral-Derivative (PID) control loop, with tunable parameters. You should begin by implementing and testing this control loop in a more accessible environment, such as your favorite desktop IDE or toolchain.

PID controllers have been or will be covered in the lecture, and lecture slide set is or will shortly be available from the course calendar. In the diagram in Figure 4, \( y(t) \) would represent the output of a feedback sensor, and \( u(t) \) would represent the desired setpoint for that sensor value (where you’d like it to end up after the controller executes for a while). In this type of controller, the error term (labeled \( e(t) \), the difference between feedback sensor output and setpoint) is treated to (1) a simple amplification process, (2) an integration process, and (3) a differentiation process. The results of these three processes are linearly combined and used to control the actuator in the system (motor, etc). "Tuning" the PID controller consists of adjusting the coefficients, also called gains, of the P, I, and D terms (\( K_P \), \( K_I \), and \( K_D \)).

![PID control loop diagram](image)

**Figure 4 – A PID control loop diagram. Courtesy Wikimedia Commons**

**PID Controller Parameters:**

- **Setpoint**
  Where the feedback sensor output value should ultimately end up after enough iterations of the control loop.

- **Loop iteration frequency (dt)**
  How often the control loop is run. Faster loops will generally perform better, but obviously chew up more CPU cycles. You will eventually need to experiment with iteration frequency in order to maximize performance without using all the CPU.

- **Proportional term gain**
- **Integral term gain**
- **Derivative term gain**

The output of this loop will eventually control motors on the blimp. In order to develop and test your PID controller on your desktop computer, you may choose to write a simple testbench with a “blimp simulator” (using Euler method and your simple physics model of the blimp).

**GOALS:**

- Develop and verify (ideally via a simple simulation) your PID control algorithm using a desktop programming environment.

**HINTS:**

- Keep in mind that floating-point math is very expensive on the MSP430, and you should try to keep it to a minimum.
The vertical infrared proximity sensor may be used to estimate the height of the blimp from the floor, and so can be used in the feedback loop for an autonomous altitude control system. This control loop should be incorporated into the blimp firmware, and should take arguments ultimately passed down from the client-side GUI.

**NOTE:** When tested prior to the lab, the IR sensor was capable of accurately detecting the floor with 1 - 2 inch resolution from a distance of maybe 3 feet, with higher precision at closer distances. However, varying floor surfaces will cause the sensor to respond differently as the blimp moves around. You may want to build a simple test rig to keep your blimp in the same spot during development.

**GOALS:**

- Implement the “Motion Controller” task outlined in Phase 1 using the PID controller from Part 8.
- Allow this task to control the motors, and to poll the IR proximity sensors.
- Allow the PID controller setpoint to be changed from the client-side application via the Set(U) command
- Allow the P,I, and D term gains to be tuned from the client-side application via the Tune(P,I,D) command

**HINTS:**

- You may need to make use of a dedicated timer ISR in your PID controller in order to get consistent timing.