Chapter 4: Propulsion Design and Actuator Interface

# Mini Abstract

1-2 paragraph chapter description. Should generally go over contents, expectations, and results. Abstracts are usually the last part of something to be written out since it is a summary of the article, but we can use them hear to help flesh out our ideas a bit for how to structure. Final abstract should be overhauled at the end of the chapter though, the chapter dictates the abstract, not the other way around.

# 

# Chapter Outline

Develop the chapter outline here. Should become very detailed and broken down to paragraph level. Remember, if we invest time and effort into making a detailed outline, the actual writing will be far easier since we understand the flow and structure before we lay out the details. Before even writing a subsection, take the time to outline that subsection in the chapter outline. A lot of writing is in the layout. Remember to update this chapter in the Master Outline file so we can all keep track of the full outline of the report, it's large so breaking it up this way should help everyone keep track of each other's ideas and work.

# Chapter 4 Draft

In this chapter we first discuss each of the major parts of the propulsion system and its interface, the specifications they were designed to meet, and the reasoning behind those specifications. Then the process of implementing the interface. Finally the process of verifying these parts and the results of the verification will be analysed.

## 4.1 Parts Selection of the Propulsion System

In order to control the barone, a propulsion system capable of meeting STR 2.0.0, Drone Speed, STR 3.0.0, Remote Control, and STR 4.0.0, Autonomous, needed to be created. In order to steer the drone effectively with its buoyant moment created by the helium lift bag the propulsion system picture in figure 4.1 was designed. For more information about the buoyant moment see section 2.2. The propulsion system consisted of a motor holding a propeller at the end of a shaft connected to a servo. This servo would turn the motor in order to change the direction of force created by the propulsion system. The motor is also connected to an ESC housed in the gondola of the Barone. This ESC sends the necessary signal to the motors to control their speeds. Both the ESCs and the servos are also connected to a microcontroller also housed in the gondola; the microcontroller provides output signals to both the ESCs and Servos based on input from the remote control transceiver. The output signal from the microcontroller controls the angle of the servo as well as the signal the ESC is sending the motor. In this section the selection process and design decisions around the parts in the propulsion system will be discussed.

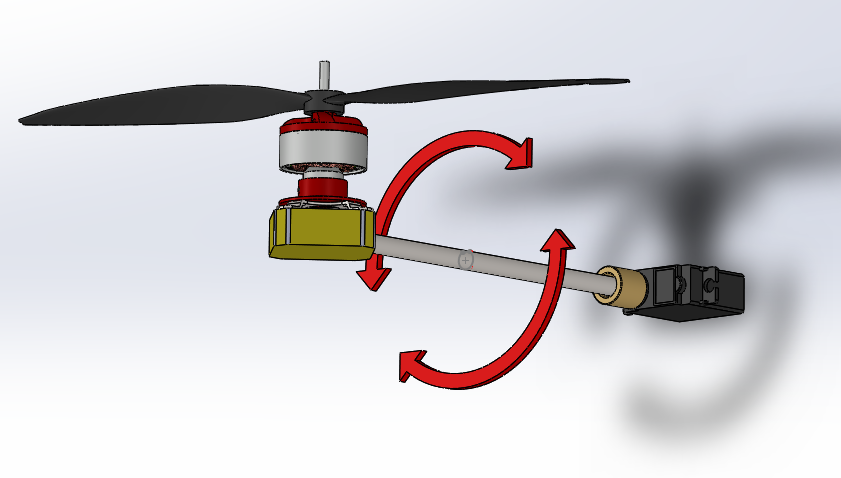


Fig.4.1 The Barone’s Propulsion System

4.1.1 Actuator selection

First a 9” propeller was chosen, since the energy the motor needs to put into spinning a larger propeller to produce the same amount of thrust as a smaller one is significantly less, and thus larger propellers are more efficient than smaller ones. This is because the kinetic energy of a rotation object is equal to [16]Additionally kinetic energy of the displaced air can be represented by [14]. We choose not to go above 9” however in order to keep the weight of the propeller relatively low. Both to meet STR 1.2.0, Weight, and reduce the force from the lever arm on the envelope, to learn more about the envelope see Chapter 3.

Next a motor was chosen based on the required throttle needed. First the required RPM needed from the motors was calculated. Since the maximum expected weight of the system was to be 5N, this was the required minimum force all four propellers needed to provide in order to get the system airborne. Using the Force equation for a propeller (4.1)where ⍴ is density, n is RPM, D is Diameter of Propeller, CT is the Thrust Coefficient, and J is Advanced Ratio[19]. CT was found using software from aerodynamics4students.com[20] suggested to us by Professor Gabrial Elkaim. Equation 4.1 was placed into a Matlab simulation the result of which is shown in figure 4.2 and it can be seen that in order for our propeller to produce 1.25N of thrust they must be spinning at 7400 RPM.

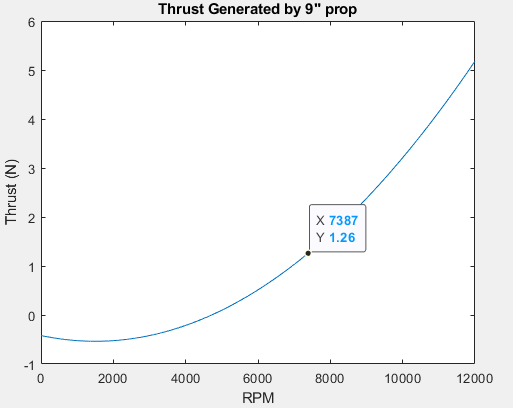


Fig.4.2 Matlab Graph of RPM vs Thrust of Our Propeller

Now that the required RPM was known a motor could be selected. A motor with a low Kv was chosen as lower KV motors have higher torque ratings and can spin larger loads with less RPM loss. Since our propeller was expected to weigh around 10g this made sense. We then needed to see that our motor would not hit its stall current while spinning the propeller at the required 7400 RPM. The expected torque needed from the motor to reach the required RPM with the propeller was found using equation (4.2)where ⍴ is density, n is RPM, D is Diameter of Propeller,CQ is the Torque Coefficient[19]. CQ was found using the same software as CT in the previous section [4]. Equation 4.2 was used to find that in order to spin our propellers at 7400, 77.43Nm of torque would be required from the motor. This torque was multiplied by the KT rating of the motor which represents the Nm of torque it can provide per amp of current drawn. The SK3 2822-1275kv Brushless motor was selected as based on itsKT rating. The expected current draw of the motor with the propeller was 0.6A at 7400 RPM with our propeller well below its 8A stall current. Additionally it had a weight of only 30g, relatively low for a motor of this size, which helped meet STR 1.2.0, Weight, as well STR 2.0.0, Drone Speed by providing the required force in order to move our system.

Racerstar RS20A BLheli\_S 4-in-1 ESC was the ESC chosen to control our motor. The Racerstar 4 in 1 ESC was chosen as each of its 4 ESCs were rated for 20A, the same rating suggested by the motors datasheet. Additionally the 4 in 1 ESC was only 25g much lighter than 4 individual ESCs or even other 4 in 1 ESCs, this also helped to meet STR 1.2.0, Weight.

Finally the servo was selected. The servo needed to be able to provide the required torque to move the rest of the propulsion system without hitting its stall torque, Additionally the servo needed to take in positional data for more than one full rotation or 360° based on the needs of our control system. See Chapter 6 for more information on the controls system. First the required torque needed to be found. This was done by using a standard torque equation. Where l is the moment of inertia and a is the angular acceleration. Using Solidworks the moment of inertia was estimated to be 2.12kgm2 x 10-4. Then the angular acceleration of the servo was found using the power and RPM of the motor by using where l was the moment of inertia and the was the stall torque of the motor found with , using the servos max power specs the RedCon 360 Degree Digital Metal Gear HV Servo was initially chosen since it had an angular acceleration of 687. The angular acceleration gave us the expected worst case toque the servo would need to apply as 0.143NM. Since this servo had a stall torque of 0.2NM it was deemed as an acceptable choice of servo. Once the servo was received and tested it was revealed that it did not take positional data as intended and was a continuous motion servo that functioned more like a motor that takes angular velocity data rather than positional data. A new servo The RC Sail Winch Servo 25T was chosen as it had a much higher stall torque of 1NM found using the same technique as the original.This servo also was able to take positional data over multiple rotations, up to 5.5 full turns.

4.1.2 Interface Selection

In order to fulfill STR 3.1.1, RC Controller, a remote controller had to be chosen that is capable of providing all necessary commands including forward, turn, ascend, and descend. It had to control the servos and motors, a remote controller and remote controller receiver were chosen. The remote controller had to transmit at least four channels for throttle, yaw, pitch, and roll commands, and the receiver had to receive at least four channels. The remote controller chosen is a 10-channel FlySky FS-i6, while the receiver is a 6-channel FlySky FS-iA6, so at most the controller could send out ten unique commands, but the receiver could only receive six of them, but this was sufficient as we only needed a minimum of four channels for throttle, yaw, pitch and roll commands. This fulfills the requirement for the remote controller.

## 4.2 Propulsion System Implementation

4.2.1 Remote Controller and Receiver Integration

In order to meet STR 3.1.2, Software for System Response of RC Control, the inputs from the remote controller and receiver had to be integrated with the PIC32.

Each channel from the receiver was outputted as a 50Hz signal with duty cycle ranging from 1ms - 2ms depending on the tilt of the joystick on the controller, and thus could be read by the PIC32MX340F512H (referenced as PIC32 for convenience) microcontroller’s Timers and Input Capture functionality in order to read the signals from the remote controller receiver.

The PIC32’s Timers are essential in time-sensitive events that can be detected by Input Capture, and can be used to set timers and trigger interrupts when those timers go off. The most important parts of the Timer are its pre-scalar and period register, or PR. Each Timer uses the Peripheral Clock of the PIC32, which is 40MHz, in order to increment its counter, and the user can configure how many cycles of the Peripheral Clock must go by in order to increment the Timer counter by one, also known as the pre-scalar. The user can configure between 1:1 pre-scalar, which means the Timer counts along with the Peripheral Clock at 40MHz, or a max of 1:256 pre-scalar, which means the Timer only increments by one after 256 cycles of the 40MHz Peripheral Clock go by. Then the user sets what the max value of the Timer should be before it goes off, which is known as the period register. For example, if a user wanted to set a 20ms/50Hz timer, you would first need to decide what pre-scalar you would like, which determines your max PR, and then you would calculate the PR. For a 20ms/50Hz timer, with a pre-scalar of 1:256, the Timer’s PR would be (20ms / 40MHz) / 256, which would give a PR of 3125[67].

Input Capture is also one of the pieces of hardware on the PIC32, and can play a very important role in measuring the length of a signal. Essentially, an Input Capture pin tracks the state of a pin, triggers an interrupt when the pin reaches a certain condition, and “captures” the value of the Timer it is connected to. In the case of the remote controller receiver, the Input Capture pin would need to capture the value of a Timer when the signal pin from a channel went high, and then once again when it went low, and subtract the Timer values to find the total amount of time that had gone by[68].

Figure 4.3 is a sample flowchart of how a standard run with the PIC32 and remote controller receiver goes:

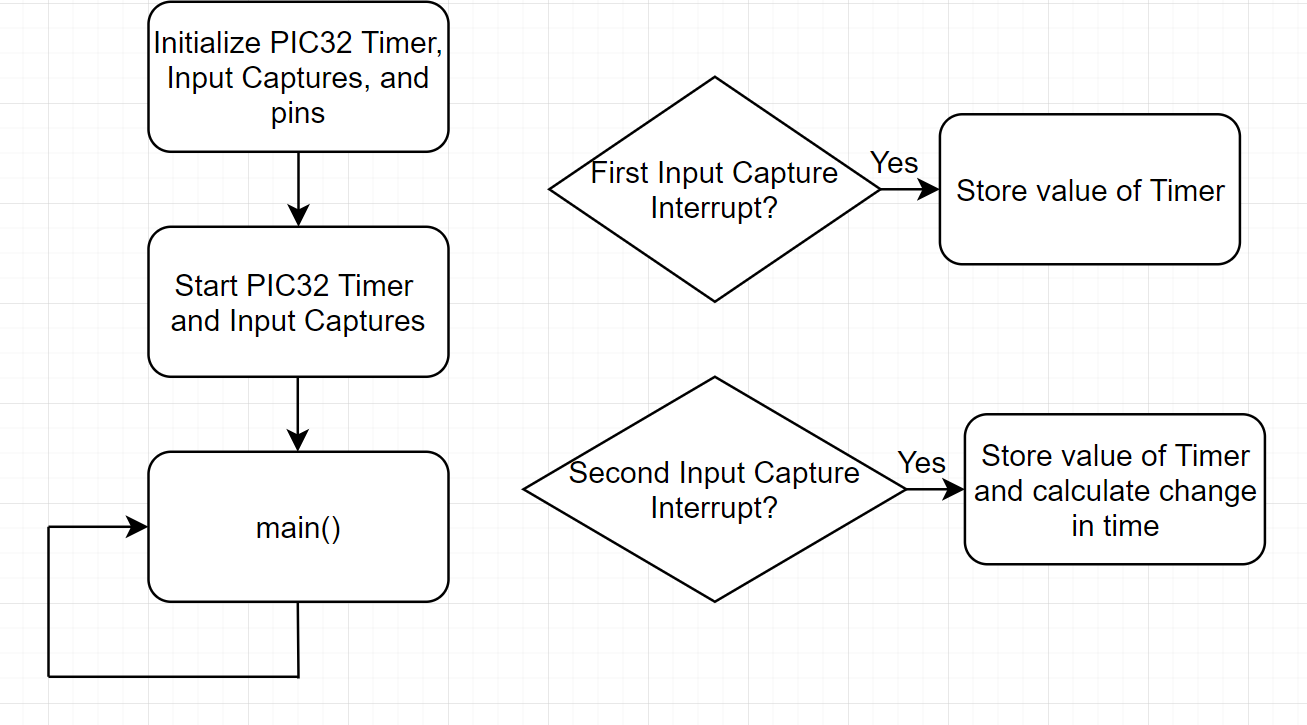


Fig. 4.3. Sample Run with PIC32 and Remote Controller Receiver

In the initialization phase, Timer 2 is initialized to be 20ms/50Hz by setting Timer 2’s pre-scalar and PR to 256 and 3125 respectively. Four Input Captures are used for the four channels from the receiver with Timer 2 as their Timer source, and configured so that each triggers an interrupt on every rising and falling edge of their respective channel pin. Each pin is configured as an input, with a starting value of 0. The Timer and Input Captures are then started, and the program enters the main loop. When a signal from the remote controller is sent to the receiver, the receiver outputs a signal with a rising edge and triggers the first Input Capture interrupt, and the user reads the value of the Timer the Input Capture captured. When the signal goes back low, the second Input Capture interrupt triggers, and the user once again reads the value of the Timer, and subtracts the first Timer value from the second to find the change in time of the signal. However, the values of the Timer are not equivalent to real-time, since they run off of the Peripheral Clock. To get this time in real-time, you need to multiply by the real-time you set the Timer to, which would be 20ms/50Hz in this case, and divide by the PR of the Timer, 3125, to get this time in milliseconds.

4.2.2 Servos, ESC, and Motors Integration

The servos and ESC, which took in a 50Hz signal and converted to a 24kHz signal for the motors, used the same input signal shown in the timing diagram in Figure 4.4:

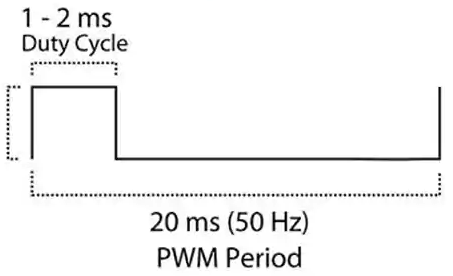


Fig. 4.4. Servo Timing Diagram

As the diagram shows, a 20ms/50Hz signal with a varying duty cycle between 1ms - 2ms needed to be produced from the microcontroller, which could have been done using the PIC32’s Output Compare functionality.

The PIC32’s Output Compare is another important piece of hardware that can be used in time-sensitive events to output a precise signal. Essentially, Output Compare works by outputting a signal set by the user, either PWM or constant high or low, when the Timer it is connected to reaches a certain value (see earlier for a description about Timers). The most important parts of the Output Compare are its mode, compare register, and compare register secondary. On initialization, the mode for Output Compare is chosen, which determines its behavior when the Timer reaches a certain value. This could be changing the signal from high to low or vice versa, or turning a duty cycle on or off. The compare register is the value the Output Compare will run until, also known as the duty cycle, and cannot be changed, after which it will perform the behavior determined by the mode, while the compare register secondary is the next value chosen by the user that will be loaded into the compare register in the next cycle. For example, for a PWM signal, the mode is chosen as being PWM, and for a 50% duty cycle, the compare register is given a value of 1562 when connected to a Timer with a PR of 3125. If the user wanted to change this duty cycle, they would change the value of the compare register secondary[69].

Figure 4.5 is a sample flowchart of how a standard run with the PIC32 and servos/ESC goes:

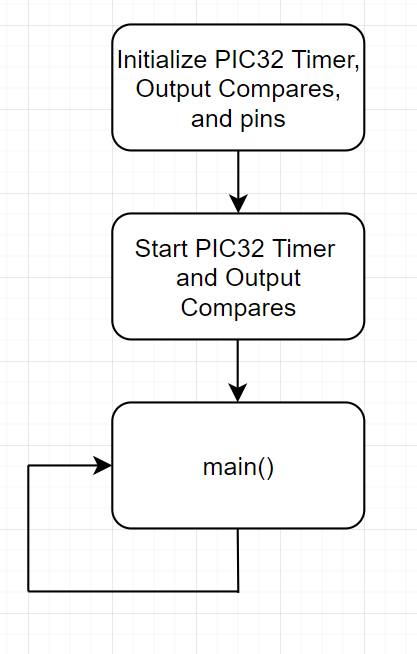


Fig. 4.5. Sample Run with PIC32 and Servos/ESC

In the initialization phase, Timer 2 is initialized to be 20ms/50Hz by setting Timer 2’s pre-scalar and PR to 256 and 3125 respectively. Two Output Compares, one for the servos and one for the ESC, are initialized to PWM mode with a starting duty cycle of 0%. The output signal pins to the servos and ESC are configured as outputs, and are given a starting value of 0. The Timer and Output Compares are then started, and the program enters the main loop. In the main loop, if the user wants to change the duty cycle of the Output Compares, they change the value of the compare register secondaries. With Timer 2 having a max value of 3125, 3125 / 4 gives 25% duty cycle, 3125 / 2 gives 50% duty cycle, and 3125 gives 100% duty cycle.

To verify the behavior of the Output Compares for the servos and ESC/motors, the Output Compares were observed on an oscilloscope as seen in Figure 4.6.

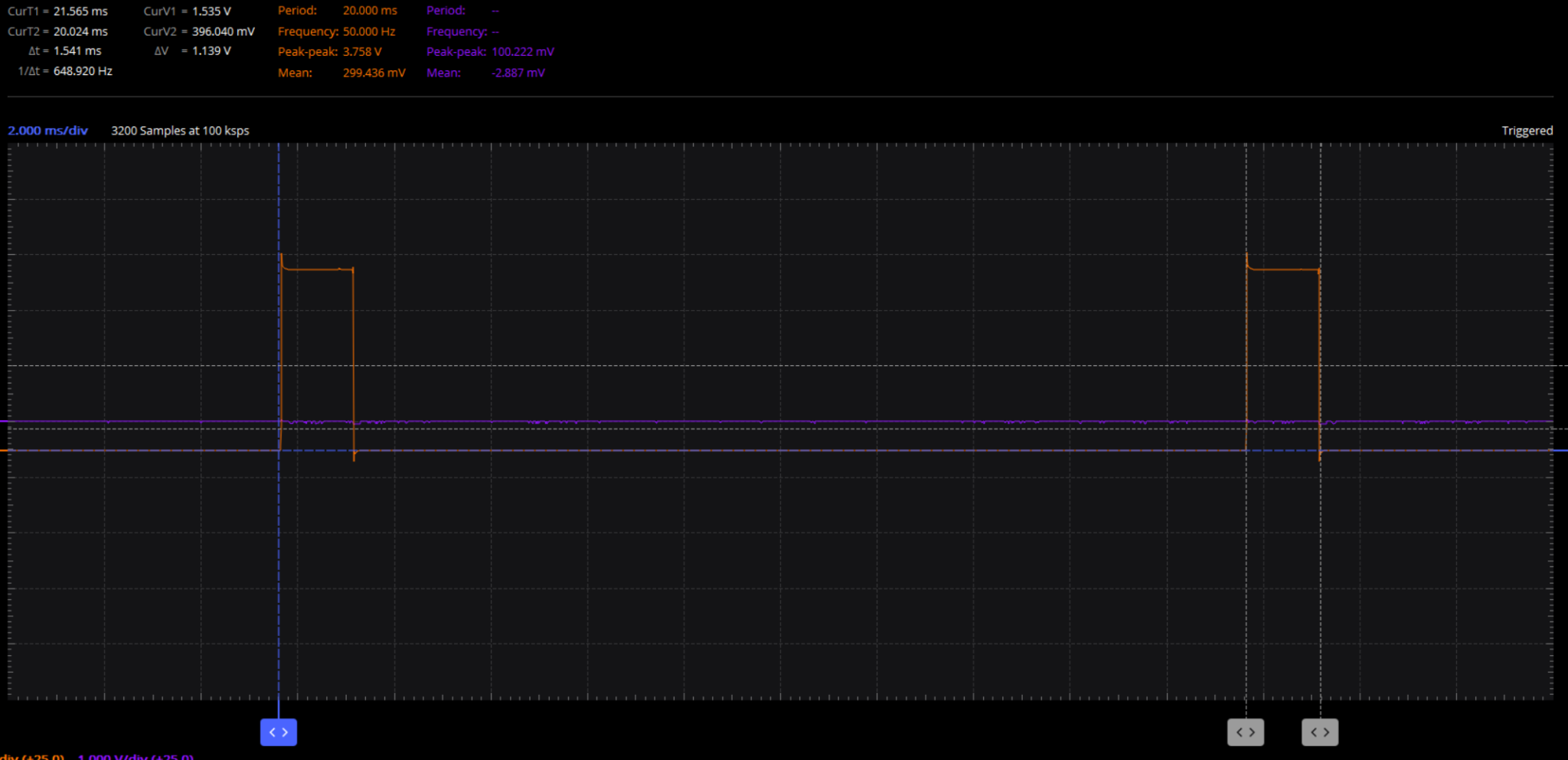


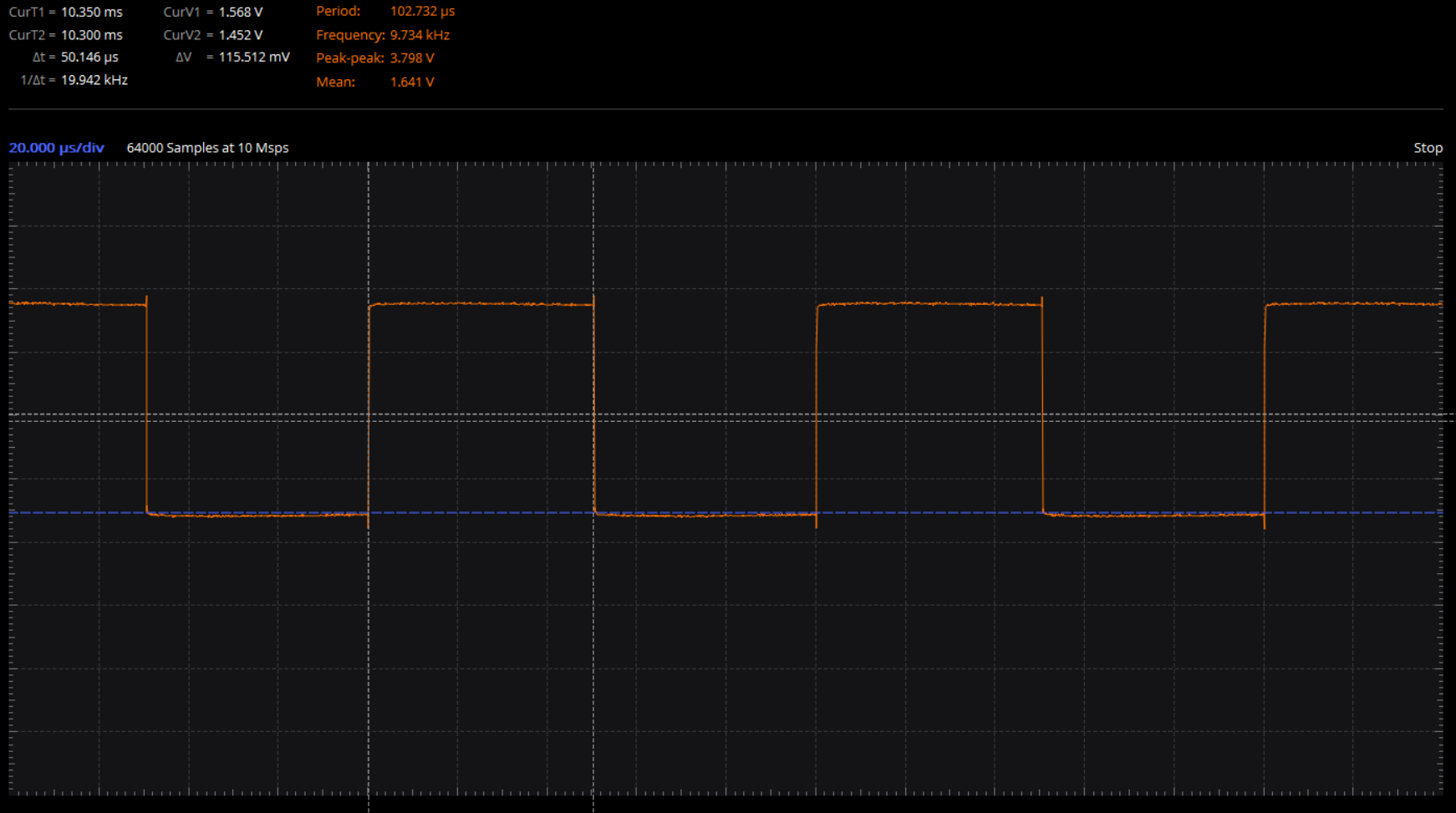
Fig. 4.6. 20ms/50Hz Frequency, 1.5ms Period Observed on an Oscilloscope

Fig 4.7. 10kHz, 50% Duty Cycle Signal Observed on an Oscilloscope

In this example, the frequency of the signal should have been 20ms/50Hz, with a duty cycle of 1.5ms, as seen in the top left corner of Figure 4.6. The signal was measured to be around 20.024ms, with a change in time of duty cycle of 1.541ms, so the Output Compares correctly outputted the right signal. This verified our implementation of Output Compare. In Figure 4.7 it can be seen that a 20kHz frequency signal with 50% duty cycle was successfully achieved when the Output Compare was configured to be half the value of the Timer it was connected to.

4.2.3 Remote Controller and Receiver, Servos, ESC, and Motors Integration

To use the remote controller receiver, servos, ESC, and motors in conjunction with each other such that the remote controller directly controls the servos and motors, the PIC32’s Input Captures and Output Compares were utilized as described in the previous section. Figure 4.7 is a sample flowchart of how a standard run with the PIC32, remote controller receiver, servos, ESC, and motors goes:

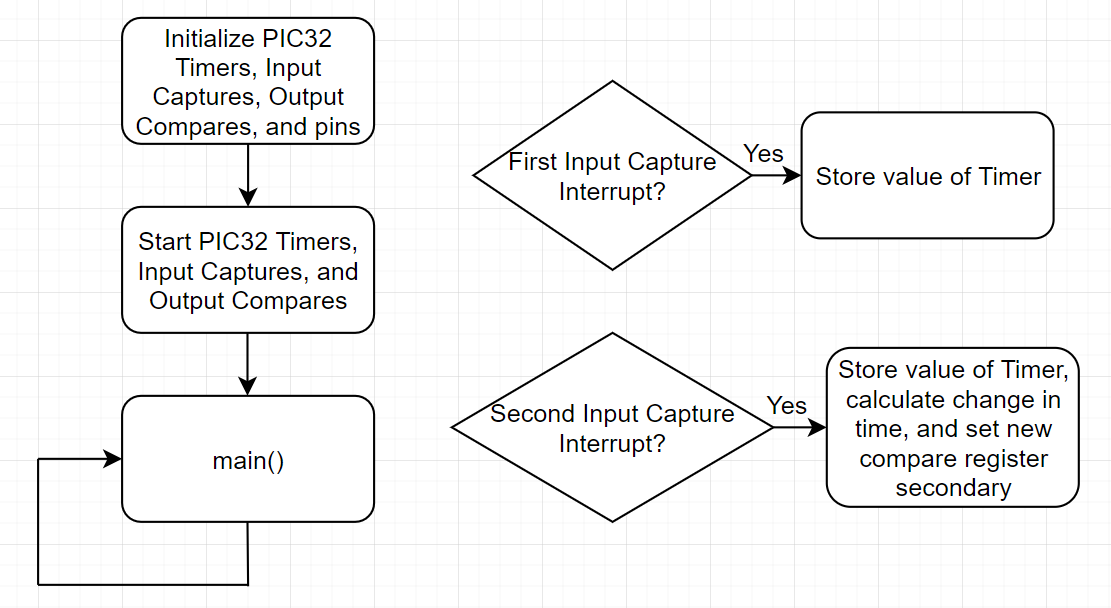


Fig. 4.7. Sample Run with PIC32 and Remote Controller, Receiver, Servos, and ESC

In the initialization phase, Timer 2 is initialized to be 20ms/50Hz by setting Timer 2’s pre-scalar and PR to 256 and 3125 respectively. Four Input Captures are used for the four channels from the receiver with Timer 2 as their Timer source, and configured so that each triggers an interrupt on every rising and falling edge of their respective channel pin. Two Output Compares, one for the servos and one for the ESC, are initialized to PWM mode with a starting duty cycle of 0%. Each Input Capture pin is configured as an input, with a starting value of 0, while each Output Compare pin to the servos and ESC are configured as outputs, and are given a starting value of 0. The Timer, Input Captures, and Output Compares are then started, and the program enters the main loop. When a signal from the remote controller is sent to the receiver, the receiver outputs a signal with a rising edge and triggers the first Input Capture interrupt, and the user reads the value of the Timer the Input Capture captured. When the signal goes back low, the second Input Capture interrupt triggers, and the user once again reads the value of the Timer, and subtracts the first Timer value from the second to find the change in time of the signal. In this same interrupt, the change in time of the signal from the Input Capture is directly used as the compare register secondary value, essentially forwarding the signal from the Input Capture pin to the Output Compare pin. For example, if in the second Input Capture interrupt, the change in time was found to be 156, this is the value one of the compare register secondaries would be assigned in order to replicate the same signal as an output.

In the current implementation, the same one signal is given for all the servos, and another same signal is given to the ESC for all motors. This is why only two Output Compares are needed, one for the servos and one for the ESC. However, the goal was for a more complicated control system where the servos would be at different angles from each other, as well as the motors having different speeds from each other.

## 4.3 Propulsion System Verification

To verify the controls system with the remote controller receiver, servos, ESC, and motors, the physical setup viewed in figure 4.8 was implemented:

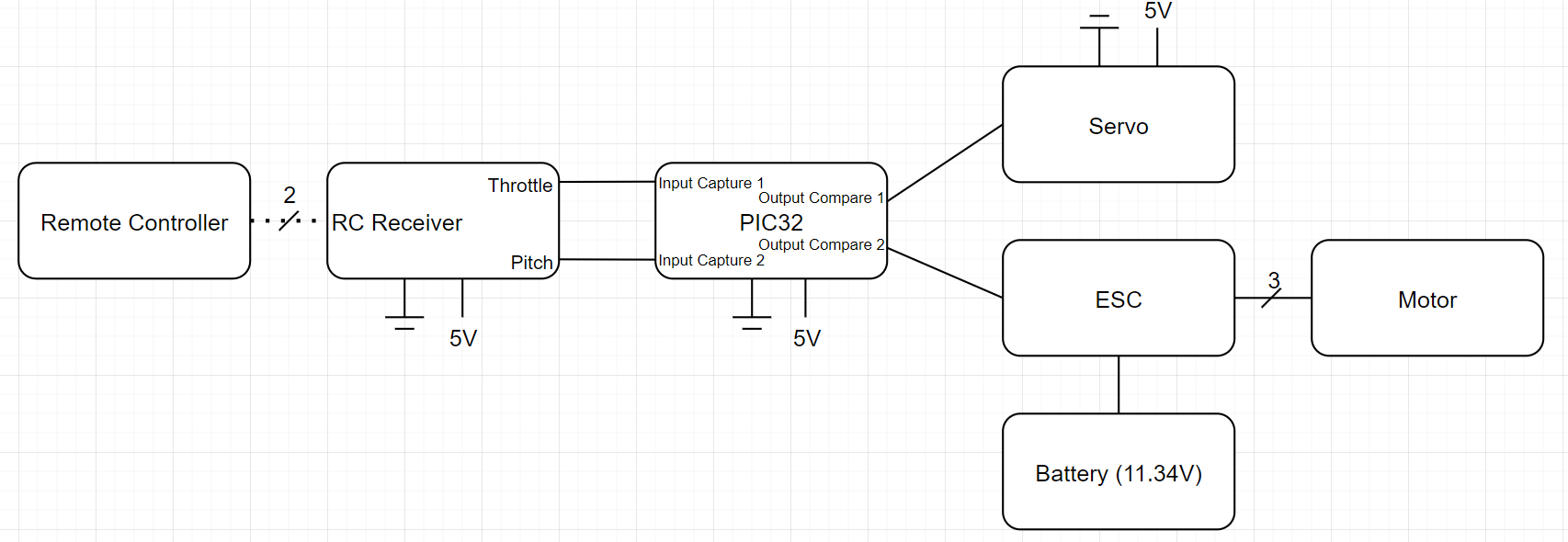


Fig. 4.8. Physical Setup with the PIC32, Remote Controller, Receiver, Servos, ESC, Motor, and Battery

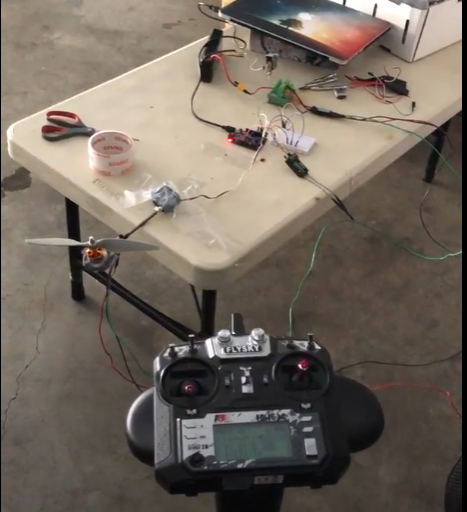


Fig. 4.9. Physical Verification of the PIC32, Remote Controller, Receiver, Servos, ESC, Motor, and Battery

With a shaft connected to the servo, and the motor with a propeller on the end of the shaft, the servo was able to rotate in either direction and the motor was able to spin at variable speeds in response to the remote controller. STR 3.1.2, Software for System Response of RC Control, has made progress in the form of controlling one servo and one motor, but is not complete until all servos and motors.

## 4.4 Conclusion

In this chapter we were successfully able to design, implement and test our propulsion system and its interface in order to attempt to meet STR 2.0.0, Drone Speed, STR 3.0.0, Remote Control, and STR 4.0.0, Autonomous. Although the final verification of the propulsion system for STR 2.0.0, Drone Speed, can be seen in Chapter 7. Here we were able to show progress on STR 3.1.2, Software for System Response of RC Control, though the implementation of Input Capture and Output Compare interrupts communication protocols between the remote controller, servos, and microcontroller, we were able to show that a single propulsion system can respond to user inputs.

# Chapter Bibliography

We do have a full bibliography that should absolutely be updated with all content here. The point of the chapter bibliography is to help keep track of citations in the chapter since the numbering may change in the full bibliography with changes and additions. This way will isolate the sources in this section so you can cite here without having to worry about it, and can use a simple find and replace on your citations to update the new numbering when we combine everything in the final report.

[19] Beard, Randal W., and Timothy W. McLain. *Small unmanned aircraft: Theory and practice*. Princeton university press, 2012

[20] *Aerodynamics for Students*, www.aerodynamics4students.com/.

[67] *PIC32 Family Reference Manual Section 14. Timers*, http://ww1.microchip.com/downloads/en/devicedoc/61105f.pdf

[68] *Section 15. Input Capture*, https://ww1.microchip.com/downloads/en/DeviceDoc/60001122G.pdf

[69] *PIC32 FRM Section 16. Output Compare*, https://ww1.microchip.com/downloads/en/DeviceDoc/61111E.pdf