Software research

**Introduction**

If you look at the mapping module research you can see that I’ve chosen to program my module on an esp32 mainly for speed reasons. The esp32 is much faster than the Arduino, but on top of that it allows me to encode in it’s own ide, ESP-IDF. In this paper I will be researching which IDE is best suitable for my paper, and into the various software libraries etc. I have to know in order to successfully make my application.

Every main function I am going to implement must be done asynchronously because it needs to be able to rotate the servo, scan the distance and send the map(on request) all at the same time. If it were to do these functions sequentially it would cause delays between the measuring, the rotating and the sending. Which will lead to an inaccurate map.

**The IDE**

The esp32 can run both Arduino IDE and it’s own IDE called ESP-IDF. Some of the advantages the Arduino IDE is very straightforward and is usable on a lot of different types of micro controllers. This means that encoding it would be much simpler than on ESP-IDF due to pre existing libraries, and it would also enable me to change microcontroller at any time without a need to change much of the code.

Although this straightforwardness also comes at a cost. Due to it’s simple design it’s also quite restrictive in what it will and will not let you do, this means that with Arduino IDE, I might not be able to use the esp32 to it’s full potential.

The ESP-IDF on the other hand offers much more and much precise control over the esp32. Which allows me to finely tweak each specific setting of the components I’m using.

For this project I have decided to go with the ESP-IDF mainly due to the fact that some essential features of the esp-32 have not yet been mapped out in the Arduino IDE (at least not in the version for the esp32) like for example i2c slave functionality. Without these functions I wouldn’t be able to finish the module.

**PWM**Servos are controlled using pwm. Pwm itself is very simple, it stands for pulse width monitor. And as the names states, it uses pulses to control the extent to which the servo will rotate. Most servos (like the one I’m using) use a pulse width ranging from 1ms to 2ms. 1 being all the way to the left and 2 being all the way to the right.

But being that this is esp-idf and not Arduino, first some steps must be manually taken to use pwm control.

First you have to initialize a chosen pin to be specifically used for pwm control, then you must configure it’s frequency, duty cycle, counter mode and duty mode.

Then to ensure the pulse is of the correct length you must make a method that converts the angle you want to the correct pulse length. The variables of this formula are based on both the maximum rotation angle it can do, and the pulse width it uses.

After these steps have been taken you are ready to control the servo.

**UART**uart stands for universal asynchronous receiver/transmitter. As the name suggests uart is used to transmit and receive data, but it’s not a communication protocol like i2c. but a physical circuit to transmit data. And it does this asynchronously which is why I’m using it for this module.

Uart has 4 “pins” RX, TX, RTS and CTS. But this module only needs to use RX and TX, which are the pins for receiving and transmitting data. And due to the fact that this is a sensor which we’ll be using, we only really need to plug in the TX pin of the sensor.

But just like with pwm, due to the fact that this is built in esp-idf, a lot of manual configurations will need to be done.

The esp32 board has multiple pin ports specifically designated for RXTX. UART0, UART1 and UART2. If the native pins for these are used there is no need to set these pins, but you can change which pins are used for RXTX by reconfiguring any of the three aforementioned ports in code. Regardless of whether or not this will be done. The parameters needed to communicate with the uart device must me set.

**XTaskCreate**XtaskCreate is an RTOS function to create a new task. A task is used to create asynchronous code loops.

The way you create a task with this method is very simple, all you have to do is fill in the parameters:

The method(either a void or static void) which you want to create a task for, a name for this task, the stack depth, pvparameters (a value that will be passed into the task, can be left blank), the priority at which the task will be executed and the pxcreatedtask (an optional parameter used to pass a handle to the created task)

**I2C**

Resources

<https://www.esp32.com/viewtopic.php?t=5669>

<https://www.instructables.com/id/Interfacing-Servo-Motor-With-ESP32/>

<http://icircuit.net/esp-idf-connected-servo-control-servo-remote-location/2101>

<https://github.com/espressif/esp-idf/blob/1e0ba341776c75515e5a3c94486487282a2d1e9b/examples/peripherals/mcpwm/mcpwm_servo_control/main/mcpwm_servo_control_example.c/>

<https://www.freertos.org/a00125.html>

<https://www.eevblog.com/forum/microcontrollers/development-environment-for-esp32-arduino-or-esp-ide/>

<https://engineering.tamu.edu/media/4247823/ds-servo-mg90s.pdf>

<https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/mcpwm.html>

<https://www.freertos.org/a00125.html>

<https://www.freertos.org/RTOS-task-states.html>