UNSA
UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

Escuela Profesional de Ingeniería Electrónica

Guía de Laboratorio de Tecnología de la Ingeniería Aeronáutica y Espacial

---

Lab 03: Programación de Modulo Giroscopio y Acelerómetro MPU6050 con FreeRTOS en Microcontrolador STM32

Alumnos

Dolmos Becerra, Uriel Frankdali
Pocohuanca Fernandez, Jeremin
Torres Cori, Josue Breithner

Profesor: Dr. Juan C. Cutipa Luque

03 de agosto del 2021

Resumen

Este informe presenta el desarrollo del uso del Módulo Acelerómetro y giroscopio MPU6050 con la placa de Desarrollo BluePill que lleva el Microcontrolador STM32F103C8T6 con Nucleo ARM Cortex M3 Usando un Sistema Operativo en Tiempo Real FreeRTOS.

Índice

1. Objetivo

- Utilizar un sistema operativo de tiempo real kernel para un microcontrolador ARM
- Entender la dinámica de los cuerpos rígidos y las tecnologías usadas en la ingeniería aeroespacial.

2. Fundamento Teórico

2.1. Modulo Desarrollo Stm32f103c8t6 Blue Pill Stm32 Cortex-m3

Ofrece una potencia, memoria flash, RAM, pines de entrada y salida, ADC, frecuencia de PWM, Resolución PWM mucho más avanzada que un Arduino.

2.1.1. Especificaciones:

- ARM 32-bit Cortex-M3 CPU Core
- Corre a una frecuencia de 72 MHz (1.25 DMIPS/MHz)
- 64 Kbytes de memoria Flash
- 20 Kbytes de SRAM
- RTC integrado y entrda de bateria de respaldo para el RTC, esto es un beneficio definitivamente ya que permite, por ejemplo, guardar datos con timestamp sin usar placas externas.
- Modo Sleep, Stop y Standby
- 26 entradas y salidas digitales, la mayoria tolerantes a 5V
- Interrupciones en todas las I/O
- 2 conversores A/D de 12-bit de 1 us, (10 entradas analogicas)
- 7 temporizadores
- 2 interfaces I2C
- 3 interfaces USART
- 2 interfaces SPI
- Interface CAN
- Micro USB para alimentacion de la placa y comunicaciones
- Se puede debuguear con SWD (con ST-Link V2)
- Dimensiones: 53 x 22 mm

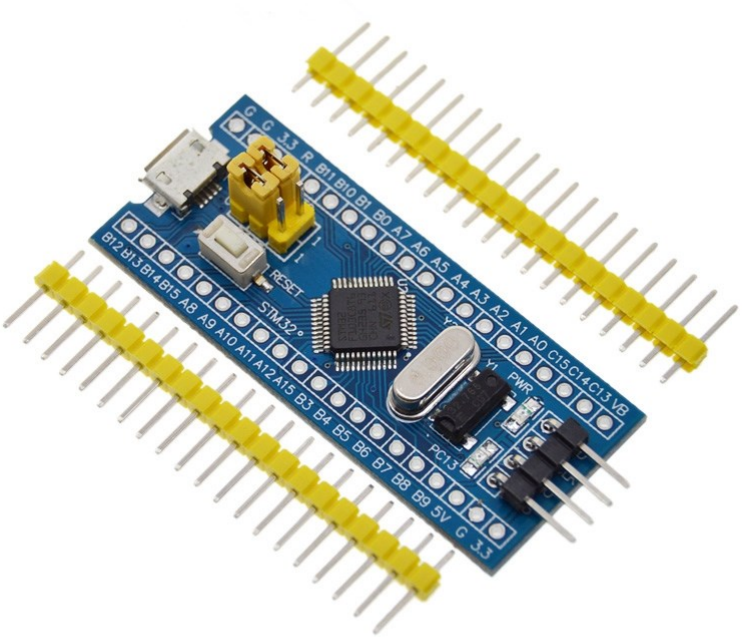

Figura 1: Modulo Desarrollo Stm32f103c8t6 Blue Pill Stm32 Cortex-m3

2.2. Módulo Acelerómetro y giroscopio MPU6050

2.2.1. Descripción

EL MPU6050 es una unidad de medición inercial o IMU (Inertial Measurment Units) de 6 grados de libertad (DoF) pues combina un acelerómetro de 3 ejes y un giroscopio de 3 ejes. Este sensor es muy utilizado en navegación, goniometría, estabilización, etc. EL módulo Acelerómetro MPU tiene un giroscopio de tres ejes con el que podemos medir velocidad angular y un acelerómetro también de 3 ejes con el que medimos los componentes X, Y y Z de la aceleración.

La dirección de los ejes está indicado en el módulo el cual hay que tener en cuenta para no equivocarnos en el signo de las aceleraciones.

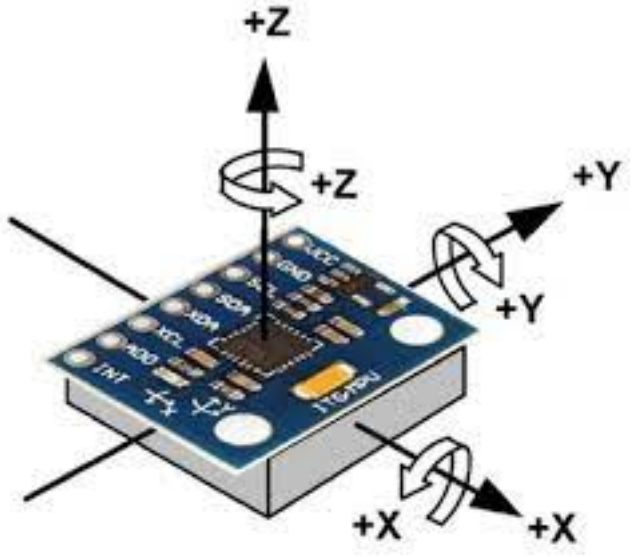

Figura 2: Módulo Acelerómetro y giroscopio MPU6050

2.3. FT232rl

2.3.1. Descripción

Simple placa conversora de série a USB con el conocido chip FT232RL de FTD, junto con dos diodos led indicadores de actividad. Se puede utilizar para alimentar y programar directamente una placa Arduino Pro (ver productos relacionados) o cualquier otro dispositivo que tenga un puerto UART TTL. Éste es el nuevo modelo que viene por defecto configurado para 5V pero dispone de un jumper en la parte posterior y se puede utilizar para 3,3V cortando la pequeña pista y soldando un pequeño punto de estaño en la parte serigrafiada como 3,3V.

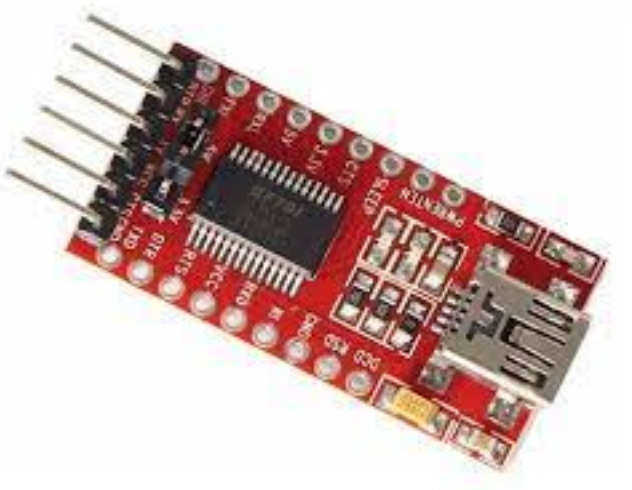

Figura 3: Ft232rl Conversor Ftdi Usb Ttl 3.3v 5v

tl

3. Materiales y Equipamientos

- Computador Laptop.
- Placa Blue Pill
- MPU6050
- raspberry FT232rl.
- interface STM32CubeIDE.

4. Procedimientos

Los procedimientos de la experiencia de Laboratorio son:

1. Entorno de Desarrollo
   El entorno elegido para programar nuestro microcontrolador STM32 es STM32CubeIDE[1]

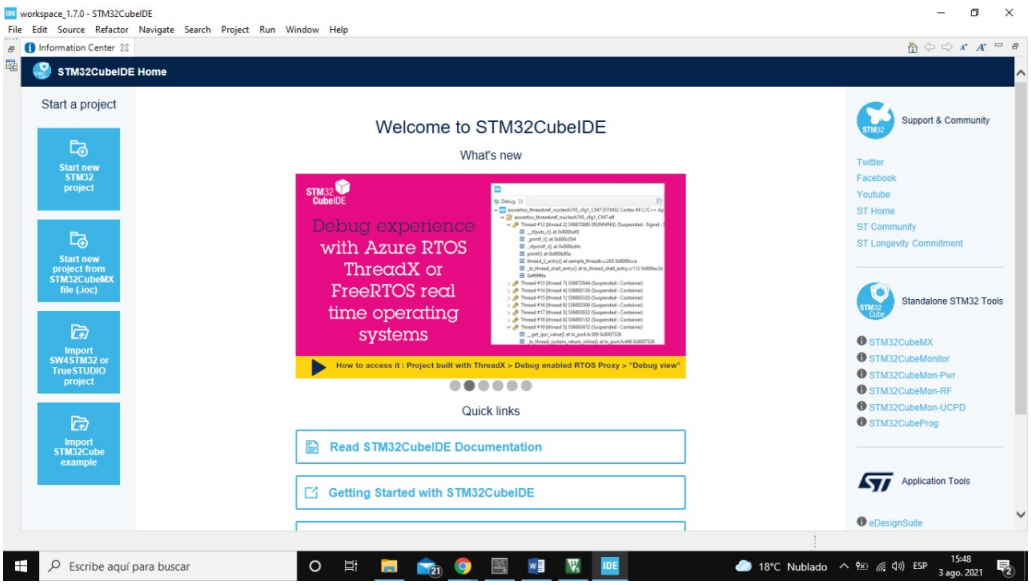

Figura 4: Pantalla Principal del IDE

Creamos un nuevo proyecto donde seleccionamos el microcontrolador STM32F103C8T6

[1]Entorno de desarrollo de sistemas operativos múltiples que forma parte del ecosistema de software STM32Cube`https://www.st.com/en/development-tools/stm32cubeide.html`

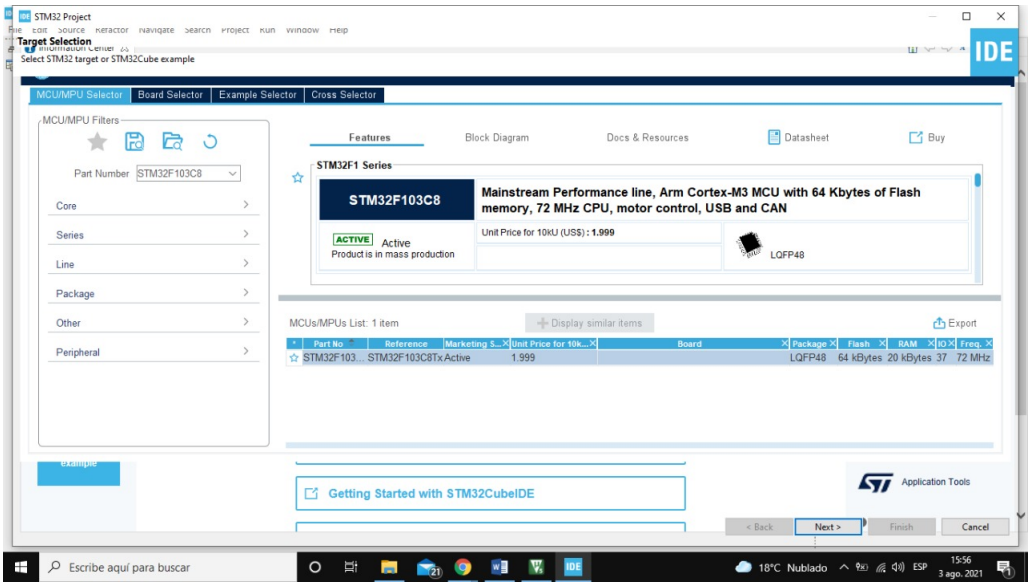

Figura 5: Selección del microcontrolador a usar

Configuramos las interfaces a utilizar y la frecuencia de reloj

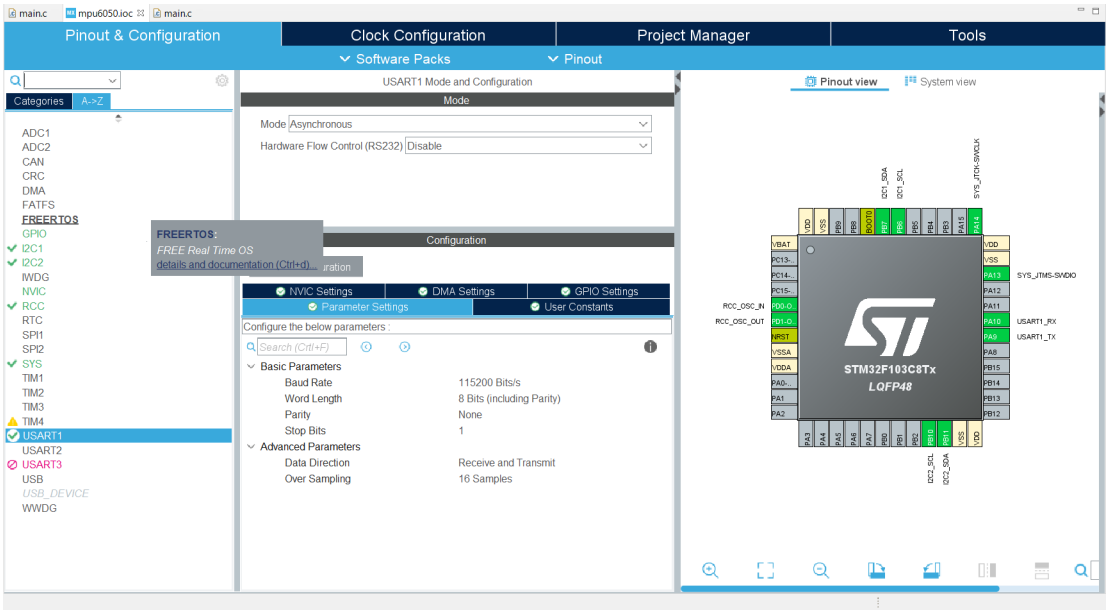

Figura 6: Selección de Interfaces y puertos

Guardamos la configuración y abrimos el archivo "main.C"que fue creado Editamos el archivo main.c con el codigo necesario para hacer funcionar nuestro microcontrolador con el modulo acelerómetro y compilamos. Se generara un archivo binario que tendremos que flashear a nuestro microcontrolador

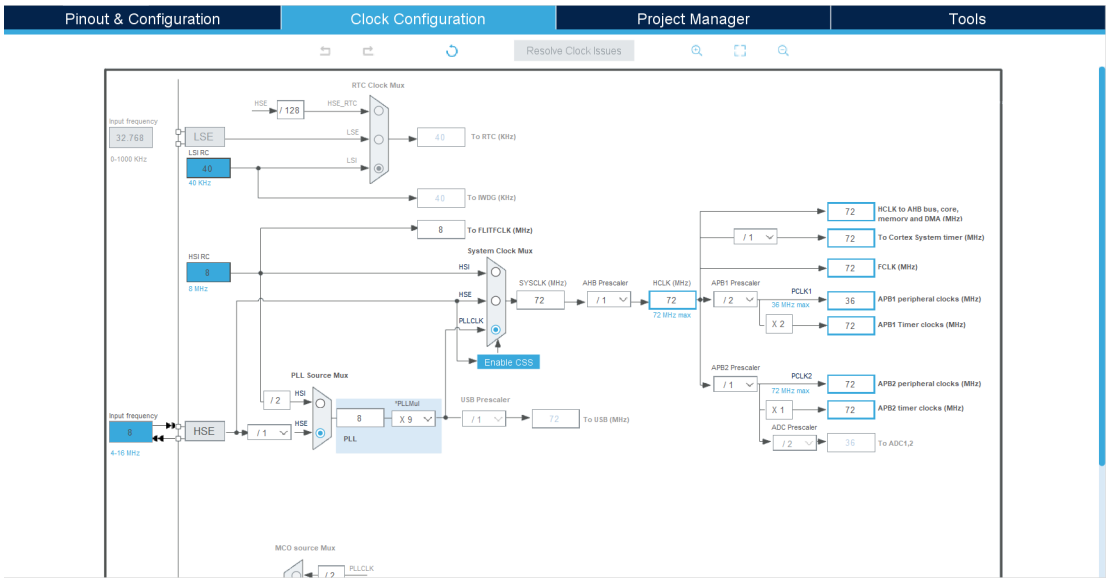

Figura 7: Configuración de Reloj

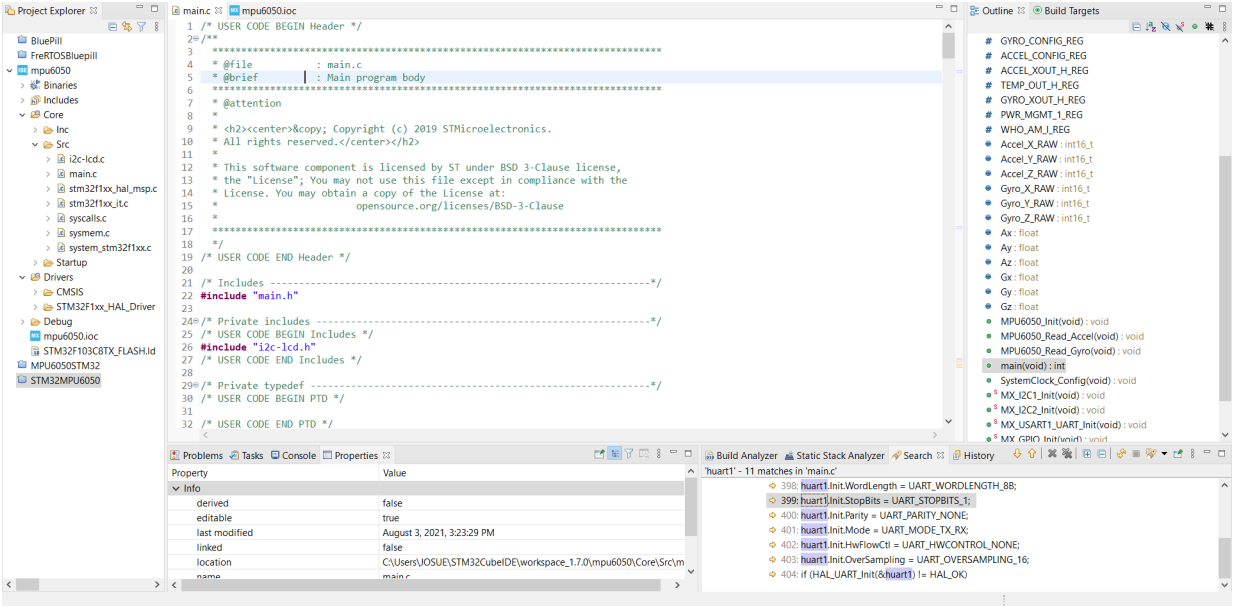

Figura 8: Codigo de nuestro Proyecto

En este caso Estamos usando unas librerías de MPU6050 para facilitar la comunicacion I2C y conseguir los datos de nuestro moduo giroscopio.

```
/* USER CODE BEGIN Header */
/**

  ******************************************************************************

  * @file           : main.c
  * @brief          : Main program body

  ******************************************************************************

  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.</center></h2>
```

```
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "cmsis_os.h"
#include "TJ_MPU6050.h"
#include "stdio.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;

UART_HandleTypeDef huart1;

osThreadId defaultTaskHandle;
osThreadId myTask02Handle;
/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
```

```
59 static void MX_USART1_UART_Init(void);
60 void StartDefaultTask(void const * argument);
61 void StartTask02(void const * argument);
62
63 /* USER CODE BEGIN PFP */
64 RawData_Def myAccelRaw, myGyroRaw;
65 ScaledData_Def myAccelScaled, myGyroScaled;
66 /* USER CODE END PFP */
67
68 /* Private user code
      ---------------------------------------------------------*/
69 /* USER CODE BEGIN 0 */
70 uint8_t buf[32] = {'\0'};
71 /* USER CODE END 0 */
72
73 /**
74   * @brief  The application entry point.
75   * @retval int
76   */
77 int main(void)
78 {
79   /* USER CODE BEGIN 1 */
80
81   /* USER CODE END 1 */
82
83   /* MCU Configuration
       ---------------------------------------------------------*/
84   MPU_ConfigTypeDef myMpuConfig;
85   /* Reset of all peripherals, Initializes the Flash interface and
      the Systick. */
86   HAL_Init();
87
88   /* USER CODE BEGIN Init */
89
90   /* USER CODE END Init */
91
92   /* Configure the system clock */
93   SystemClock_Config();
94
95   /* USER CODE BEGIN SysInit */
96
97   /* USER CODE END SysInit */
98
99   /* Initialize all configured peripherals */
100  MX_GPIO_Init();
101  MX_I2C1_Init();
102  MX_USART1_UART_Init();
103  /* USER CODE BEGIN 2 */
104
105  /* USER CODE END 2 */
106
107  /* USER CODE BEGIN RTOS_MUTEX */
108  /* add mutexes, ... */
109  /* USER CODE END RTOS_MUTEX */
110
111  /* USER CODE BEGIN RTOS_SEMAPHORES */
112  /* add semaphores, ... */
113  /* USER CODE END RTOS_SEMAPHORES */
114
115  /* USER CODE BEGIN RTOS_TIMERS */
```

```
116   /* start timers, add new ones, ... */
117   /* USER CODE END RTOS_TIMERS */
118
119   /* USER CODE BEGIN RTOS_QUEUES */
120   /* add queues, ... */
121   /* USER CODE END RTOS_QUEUES */
122
123   /* Create the thread(s) */
124   /* definition and creation of defaultTask */
125   osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0,
        128);
126   defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);
127
128   /* definition and creation of myTask02 */
129   osThreadDef(myTask02, StartTask02, osPriorityNormal, 0, 128);
130   myTask02Handle = osThreadCreate(osThread(myTask02), NULL);
131
132   /* USER CODE BEGIN RTOS_THREADS */
133   /* add threads, ... */
134   /* USER CODE END RTOS_THREADS */
135   MPU6050_Init(&hi2c1);
136     //2. Configure Accel and Gyro parameters
137     myMpuConfig.Accel_Full_Scale = AFS_SEL_2g;
138     myMpuConfig.ClockSource = Internal_8MHz;
139     myMpuConfig.CONFIG_DLPF = DLPF_184A_188G_Hz;
140     myMpuConfig.Gyro_Full_Scale = FS_SEL_500;
141     myMpuConfig.Sleep_Mode_Bit = 0;  //1: sleep mode, 0: normal mode
142     MPU6050_Config(&myMpuConfig);
143   /* Start scheduler */
144   osKernelStart();
145
146   /* We should never get here as control is now taken by the
        scheduler */
147   /* Infinite loop */
148   /* USER CODE BEGIN WHILE */
149   while (1)
150   {
151     /* USER CODE END WHILE */
152
153     /* USER CODE BEGIN 3 */
154   }
155   /* USER CODE END 3 */
156 }
157
158 /**
159   * @brief System Clock Configuration
160   * @retval None
161   */
162 void SystemClock_Config(void)
163 {
164   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
165   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
166
167   /** Initializes the RCC Oscillators according to the specified
        parameters
168   * in the RCC_OscInitTypeDef structure.
169   */
170   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
171   RCC_OscInitStruct.HSEState = RCC_HSE_ON;
172   RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
```

```
173   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
174   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
175   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
176   RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
177   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
178   {
179     Error_Handler();
180   }
181   /** Initializes the CPU, AHB and APB buses clocks
182   */
183   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|
      RCC_CLOCKTYPE_SYSCLK
184                                 |RCC_CLOCKTYPE_PCLK1|
      RCC_CLOCKTYPE_PCLK2;
185   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
186   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
187   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
188   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
189
190   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) !=
      HAL_OK)
191   {
192     Error_Handler();
193   }
194 }
195
196 /**
197   * @brief I2C1 Initialization Function
198   * @param None
199   * @retval None
200   */
201 static void MX_I2C1_Init(void)
202 {
203
204   /* USER CODE BEGIN I2C1_Init 0 */
205
206   /* USER CODE END I2C1_Init 0 */
207
208   /* USER CODE BEGIN I2C1_Init 1 */
209
210   /* USER CODE END I2C1_Init 1 */
211   hi2c1.Instance = I2C1;
212   hi2c1.Init.ClockSpeed = 100000;
213   hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
214   hi2c1.Init.OwnAddress1 = 0;
215   hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
216   hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
217   hi2c1.Init.OwnAddress2 = 0;
218   hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
219   hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
220   if (HAL_I2C_Init(&hi2c1) != HAL_OK)
221   {
222     Error_Handler();
223   }
224   /* USER CODE BEGIN I2C1_Init 2 */
225
226   /* USER CODE END I2C1_Init 2 */
227
228 }
229
```

```
/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13|GPIO_PIN_14, GPIO_PIN_RESET);

  /*Configure GPIO pins : PC13 PC14 */
  GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

}

```

```
/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/* USER CODE BEGIN Header_StartDefaultTask */
/**
  * @brief  Function implementing the defaultTask thread.
  * @param  argument: Not used
  * @retval None
  */
/* USER CODE END Header_StartDefaultTask */
void StartDefaultTask(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {

  MPU6050_Get_Accel_RawData(&myAccelRaw);
  MPU6050_Get_Gyro_RawData(&myGyroRaw);
  //HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
    //osDelay(20);
  osDelay(1000);
  }
  /* USER CODE END 5 */
}

/* USER CODE BEGIN Header_StartTask02 */
/**
* @brief Function implementing the myTask02 thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_StartTask02 */
void StartTask02(void const * argument)
{
  /* USER CODE BEGIN StartTask02 */
  /* Infinite loop */
  for(;;)
  {
    MPU6050_Get_Accel_Scale(&myAccelScaled);
    MPU6050_Get_Gyro_Scale(&myGyroScaled);
    sprintf (buf, "Ax=%.2f \n", myAccelScaled.x);
    HAL_UART_Transmit(&huart1, buf, sizeof(buf), 100);

    sprintf (buf, "Ay=%.2f \n", myAccelScaled.y);
    HAL_UART_Transmit(&huart1, buf, sizeof(buf), 100);

    sprintf (buf, "Az=%.2f \n", myAccelScaled.z);
    HAL_UART_Transmit(&huart1, buf, sizeof(buf), 100);

    sprintf (buf, "Gx=%.2f \n", myGyroScaled.x);
    HAL_UART_Transmit(&huart1, buf, sizeof(buf), 100);

    sprintf (buf, "Gy=%.2f \n", myGyroScaled.y);
    HAL_UART_Transmit(&huart1, buf, sizeof(buf), 100);

    sprintf (buf, "Gz=%.2f \n", myGyroScaled.z);
    HAL_UART_Transmit(&huart1, buf, sizeof(buf), 100);
    //HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_14);
```

```
350     //osDelay(200);
351     osDelay(1000);
352   }
353   /* USER CODE END StartTask02 */
354 }
355
356 /**
357   * @brief  Period elapsed callback in non blocking mode
358   * @note   This function is called  when TIM1 interrupt took place,
        inside
359   * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to
        increment
360   * a global variable "uwTick" used as application time base.
361   * @param  htim : TIM handle
362   * @retval None
363   */
364 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
365 {
366   /* USER CODE BEGIN Callback 0 */
367
368   /* USER CODE END Callback 0 */
369   if (htim->Instance == TIM1) {
370     HAL_IncTick();
371   }
372   /* USER CODE BEGIN Callback 1 */
373
374   /* USER CODE END Callback 1 */
375 }
376
377 /**
378   * @brief  This function is executed in case of error occurrence.
379   * @retval None
380   */
381 void Error_Handler(void)
382 {
383   /* USER CODE BEGIN Error_Handler_Debug */
384   /* User can add his own implementation to report the HAL error
      return state */
385   __disable_irq();
386   while (1)
387   {
388   }
389   /* USER CODE END Error_Handler_Debug */
390 }
391
392 #ifdef  USE_FULL_ASSERT
393 /**
394   * @brief  Reports the name of the source file and the source line
      number
395   *         where the assert_param error has occurred.
396   * @param  file: pointer to the source file name
397   * @param  line: assert_param error line source number
398   * @retval None
399   */
400 void assert_failed(uint8_t *file, uint32_t line)
401 {
402   /* USER CODE BEGIN 6 */
403   /* User can add his own implementation to report the file name and
       line number,
404     ex: printf("Wrong parameters value: file  %s on line  %d\r\n",
```

```
    file, line) */
405   /* USER CODE END 6 */
406 }
407 #endif /* USE_FULL_ASSERT */
408
409 /************************ (C) COPYRIGHT STMicroelectronics *****END
    OF FILE****/
```

2. Flashear el programa del microcontrolador
   El STM32F103C8T6 se programa mediante ST-Link V2. Como no tenemos esta herramienta, tenemos que grabar el archivo binario mediante el Puerto USART1. En nuestro caso se ha presentado conflictos para flashear con el modulo FT232RL por lo que usamos el puerto UART de la RaspberryPi. Para ello Seguiremos la guia de la Siguiente Pagina `https://siliconjunction.wordpress.com/2017/03/21/flashing-the-stm32f-board-using-a-raspberry-pi-3/` donde instalamos la utilidad stm32flash y conectamos el microcontrolador al raspberry Pi de la siguiente manera:

   - Raspberry Pi 3.3V (pin 1) a STM32 3.3V
   - Raspberry Pi GND (pin 6) a STM32 GND
   - Raspberry Pi TX (pin 8) a STM32 RX (pin A10)
   - Raspberry Pi RX (pin 10) a STM32 TX (pin A9)
   - Establecemos el jumper STM32 BOOT0 en 1

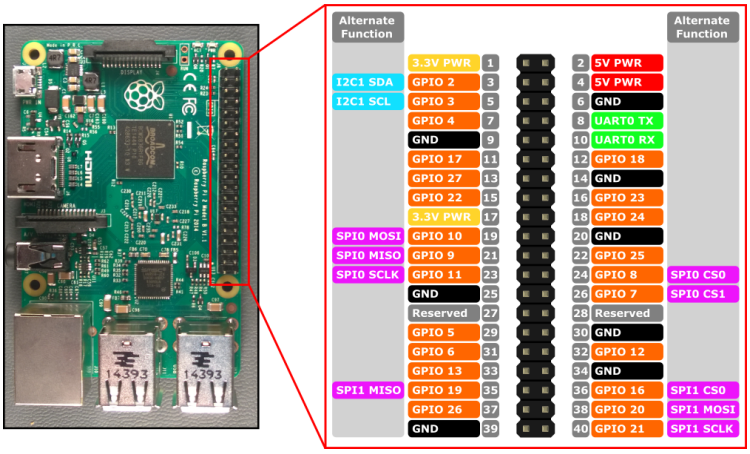

Figura 9: Pin Out de la Raspberry Pi

Grabamos el binario, Establecemos el jumper STM32 BOOT0 en 0 y reiniciamos el microcontrolador para iniciar el programa.

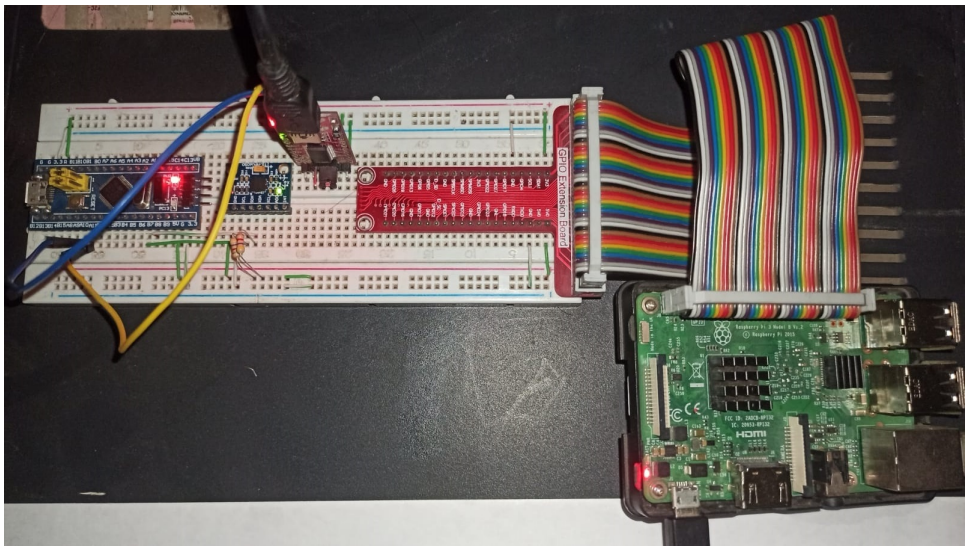

Figura 10: Circuito en Protoboard

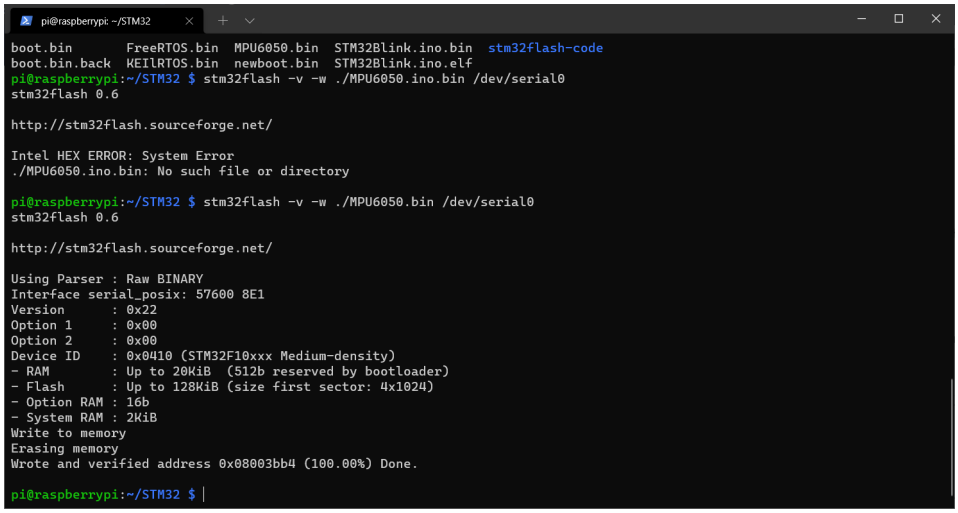

Figura 11: Herramienta Stm32flash corriendo en la Raspberry Pi

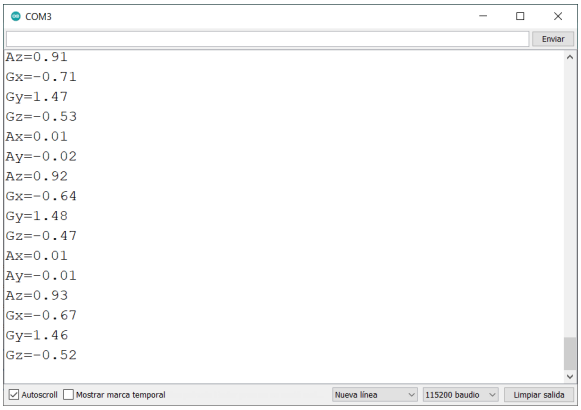

Figura 12: Monitor Serie de Arduino Mostrando los valores del giroscopio

5. Conclusiones

- Se logro Utilizar el Microcontrolador STM32F103C8T6 con el Modulo MP6050

- Un entorno de desarrollo como el STM32CubeiDE nos permite tener mas contro sobre nuestro microcontrolador
- Los sistemas operativos en tiempo Real nos facilitan la gestion de diferentes actividades que programemos en nuestro microcontrolador

6. Repositorio

- Github: `https://github.com/jtorrescor/Lab-03-STM32-MPU6050`.
- OverLeaf: `https://es.overleaf.com/read/hfcvfkmzczkz`.

Referencias

- Warren Gay, (2018). Beginning STM32, Developing with FreeRTOS, libopencm3 and GCC `https://oiipdf.com/download/1783`
- Conectar modulo MPU6050 con STM32 `https://controllerstech.com/how-to-interface-mpu6050-gy-521-with-`
- Depuración de STM32 con Impresión en puerto Serie UART `https://deepbluembedded.com/stm32-debugging-with-uart-serial-prin`
- Librerías STM32 `https://github.com/MYaqoobEmbedded/STM32-Tutorials`.

Rúbrica

e1: Identifica y diagnostica problemas y los prioriza de acuerdo a su impacto o relevancia.

e2: Formula soluciones coherentes y realizables usando normas y estándares apropiados.

e3: Utiliza las técnicas y metodologías de la ingeniería electrónica para plantear, analizar y resolver problemas de ingeniería.

e4: Maneja equipos e instrumentos y utiliza software especializado propio del ejercicio profesional.

La tabla 1 refleja la evaluación del estudiante respecto este informe y mediante entrevistas.

Tabla 1: Rúbrica según Resultados del Estudiante

Alumno	e1	e2	e3	e4
Dolmos Becerra, Uriel Frankdali				
Pocohuanca Fernandez, Jeremin				
Torres Cori, Josue Breithner				