Software Formalization

Year: \_2021\_ Semester: \_Fall\_\_\_ Team: \_\_6\_\_ Project:\_\_\_\_\_\_\_RevEx\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Creation Date: \_\_\_\_October 4, 2021\_\_\_\_\_\_\_\_\_\_ Last Modified: October 8, 2021

Author: \_\_\_\_\_\_\_\_Isaac Hagedorn\_\_\_\_\_\_\_\_ Email: \_\_\_\_\_\_ihagedo@purdue.ed\_\_\_\_\_

Assignment Evaluation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Score (0-5)** | **Weight** | **Points** | **Notes** |
| **Assignment-Specific Items** | | | | |
| **Third Party Software** |  | x2 |  |  |
| **Description of Components** |  | X3 |  |  |
| **Testing Plan** |  | x3 |  |  |
| **Software Component Diagram** |  | x4 |  |  |
| **Writing-Specific Items** | | | | |
| **Spelling and Grammar** |  | x2 |  |  |
| **Formatting and Citations** |  | x1 |  |  |
| **Figures and Graphs** |  | x2 |  |  |
| **Technical Writing Style** |  | x3 |  |  |
| **Total Score** |  | | |  |

5: Excellent 4: Good 3: Acceptable 2: Poor 1: Very Poor 0: Not attempted

General Comments:

*Relevant overall comments about the paper will be included here*

1.0 Utilization of Third Party Software

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **License** | **Description** | **Use** |
| x-io Technologies Fusion | GPL | “Fusion is an ANSI C99 compliment sensor fusion library for sensor arrays of gyroscopes, accelerometers, and magnetometers. Fusion was specifically developed for use with embedded systems and has been optimised for execution speed. The library includes modules for: attitude and heading reference system (AHRS) sensor fusion, gyroscope bias correction, and a tilt-compensated compass.” [2] | We will be using this library to determine the orientation vector of the upper arm relative to the shoulder. |
| Unity | Unity Personal Use | Unity [6] is a cross-platform game engine that can be used to create both 2D and 3D games as well as interactive simulations. | We will be using Unity for our VR simulation on the host computer. |
| BleWinrtDll | WTFPL | “This VisualStudio-project compiles to a C++-dll that can be imported into Unity. It wraps a part of the UWP BLE API inside a dll. The dll can be simply dropped into your Unity project and be used in the Unity Editor and the Windows standalone version.” [1] | We may be using this library to abstract calls to Microsoft’s UWP BLE API. |
| STM32L0 | BSD 3-Clause | This library provides a convenient memory map for the entire MCU as well as a few weak interrupt handlers which can be overwritten. | We will use this library for the convenient memory mapping and memory structures. |
| STM32 HAL | BSD 3-Clause | “The HAL driver layer provides a generic multi-instance simple set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks)”[5].  This library provides several drivers for various on-board peripherals and pinouts. The library includes drivers to initialize the SPI and UART communication, initiate transmissions, and handle receptions. Additionally, the library has other various functions such as sleep/clock management, interrupt handlers, and other peripheral drivers. | We will use this library for the well-established UART and SPI drivers/handlers as well as for clock management. |
| ICM-20948 | MIT | This library provides a memory map and functions for commanding/configuring the chip and reading sensor data over SPI from the gyroscope, accelerometer, and magnetometer [4]. | We will use this library to initialize the IMU and its sensors and to read data from the various on-board sensors. |

2.0 Description of Software Components

# **2.1 SPI (IMU)**

The SPI communication protocol will be used to communicate at extremely high frequencies with the IMU. This communication protocol consists of development on two separate chips, the microcontroller and the IMU. The team will have control over development for the MCU side but will use the code developed on the IMU chip as-is. Additionally, the team will use the STM HAL Driver library to help initialize the SPI communication on the MCU side. Finally, the team will re-write portions of a provided software library from Sparkfun to interface with the chip. This library includes memory mapping as well as a few skeleton functions for reading and writing to specific registers on the IMU. The main functions include:

icm20948\_init(): This function initializes the IMU and handles calibration for each of the different sensors. It returns the status of the initialization and calibration.

**read\_SPI**(unit8\_t addr): This function takes an address and initiates a read command over SPI. The return data is the return from the read command.

**write\_SPI**(uint8\_t addr, uint8\_t data): This function takes an address and data to be written to the IMu and initiates a write command. There is no return value.

**accel\_read**(axises \* ptr): This function takes a data structure that includes the X, Y, and Z axises and initiates a read command on the accelerometer for each axis. The return data is stored in the structure and values are returned by address.

**mag\_read**(axises \* ptr): This function takes a data structure that includes the X, Y, and Z axises and initiates a read command on the magnetometer for each axis. The return data is stored in the structure and values are returned by address.

**gyro\_read**(axises \* ptr): This function takes a data structure that includes the X, Y, and Z axises and initiates a read command on the gyroscope for each axis. The return data is stored in the structure and values are returned by address.

The IMU itself will provide functionality for sampling gyroscope, magnetometer, and accelerometer data along the X, Y, and Z, axis. Additionally, the IMU will provide functionality for sensor fusion. These functionalities will be used as-is from the chip manufacturer.

# **2.2 ADC (Analog potentiometer/Battery Status)**

The MCU will use several ADCs to read potentiometer data for angular feedback from the brace and to read the battery voltage for tracking the external battery status. The tem will use the ADCs provided by STMicroelectronics on the MCU and will develop drivers for initializing and sampling the ADCs in low power mode. A lookup table will also be developed for the potentiometer to help map values to specific angles.

Additionally, the team will use an ADC to characterize the voltage supply of a battery at any given moment. Like the potentiometer, a lookup table will be used to map the digital value and track the battery status at different stages of its lifetime.

# **2.3 UART**

The UART protocol will be used for communication between the MCU and the BLE chip. The development of this protocol will only be handled on the MCU side as the UART protocol for the BLE chip will be used as-is. The team will use the STM HAL Driver library to initialize the UART on the MCU side and handle transmissions through interrupts.

# **2.4 Bluetooth Low Energy (BLE) Communication**

On the MCU side the Bluetooth communication will be handled entirely by the BLE chip through its Microchip Low-Energy Data Profile (MLDP) protocol [3]. The chip will be configured via UART commands that set specific register bits in order to bond to the host and auto-advertise data via the MLDP for continuous transmission. Additionally, the chip will automatically generate an external interrupt when data is received back from the host. The development necessary will be for the initialization of the chip and communication.

On the host side, a BleTranceiver class will use Microsoft UWP BLE API (using BleWinrtDll[1]) to communicate with the wearable. The main functions of this include:

* bool **Connect**(): Scan for connectable BLE devices and attempt to subscribe to the corresponding device (including specific service and characteristic) to the MCU bluetooth module based on known configuration. Return a bool indicating success of the subscribe.
* void **HandlePacketReceived**(): Spawns a thread that polls for new packets then calls GetParsedPackets and AddDataPacket.
* void **HandlePacketSend**(uint8\_t): Send the packet containing the haptic feedback intensity value.
* void **AddSensorPacket**(byte[1]): Creates structs containing the IMU and potentiometer data from a retrieved sensor packet. This includes converting the two-byte representations of the float measurements back to floating point values. Then enqueue this struct into to a packet queue. Mutex locks will be used to prevent races when the main Unity thread calls GetSensorPacket.
* SensorSample **GetSensorPacket**(): Dequeue packet from the queue.

# **2.5 Host Computer VR Simulation**

There are 3 major components to the host computer VR simulation: sensor fusion (estimate arm position), VR scene (enabling users’ interaction with virtual objects), and calculating the haptic feedback information.

# **2.5.1 Sensor Fusion**

The host-computer will utilize x-io Technologies’ Madgwick filter implementation [2] to determine the orientation vector of the upper arm relative to the shoulder. Specifically, the roll, pitch, and yaw angles will be obtained using the FusionQuaternionToEulerAngles() and all dependent functions as in this [Fusion/ExampleAhrs.c](https://github.com/xioTechnologies/Fusion/blob/master/Examples/ExampleAhrs.c) example program [2].

As detailed in the Software Overview, a calibration process is required to estimate the shoulder, elbow, and wrist joint locations. The main functions for calibrating and obtaining the joint locations include:

* Transform **CalibrateShoulderTf**(): Instruct the user to move their arm in an arc from one specific position to another while holding the controller. Sample three points along the arc and obtain the perpendicular bisector of points one and two as well as from point two and three. The shoulder position is then the intersection of these lines. Return a Unity Transform corresponding to this 3D point.
* Tuple<float, float> **CalibrateArmLengths**(Transform): Instruct the user to fully extend their arm such that we can find the total distance from the shoulder tf to the hand (controller). Then we can get the position of the elbow using the same intersection technique used in CalibrateShoulderTf. Once we have the elbow transform and total arm length, we can get the forearm length by subtracting the upper arm length (distance from shoulder to elbow) from the total arm length.
* Tuple<Transform, Transform> **GetElbowAndWristTfs**(Transform, SensorSample, float, float): Use the shoulder position, orientation (roll, pitch, yaw) vector from the Madgwick filter, and the upper arm length to infer the elbow position. This elbow position is fused with the elbow angle received in the data packet and the forearm length to infer the wrist position. The corresponding elbow and wrist transforms are returned.

# **2.5.2 VR Scene**

Once we have the individual arm joints, we can render a virtual representation of the user’s arm in VR to enable the user to manipulate (e.g. move, pick up, or knockover) objects in VR. The main functions include:

* GameObject **RenderArm**(Transform, Transform, Transform): Create a 3D game object in the scene that represents the user’s arm using the shoulder, elbow, and wrist joint locations.
* void **RenderScene**(ConfigurationOptions): Create a 3D VR scene with game objects that the user can interact with (e.g. pull, push, lift, etc.).
* void **UpdateObjectInteractions**(All GameObjects): A wrapper on Unities particle/physics simulations to manage the interactions between the arm game object and all other scene game objects.

# **2.5.2 Haptic Feedback**

The haptic feedback intensity value will indicate how much braking force we want to apply for haptic feedback. Reiterating the haptic feedback calculations in Section 2.2 of the Software Overview, the haptic feedback calculation function as as follows:

* 3DVec **GetOrthogonalWristJointForce**(GameObject, Transform): Use the physics engine in the VR toolbox to calculate the force vector on the wrist joint using the motion of the game object the user is currently interacting with.
* uint8\_t **GetHapticFeedbackIntensity**(3DVec, float): An 8-bit integer where 255 indicates maximum haptic feedback and 0 indicates no haptic feedback. Based on the max torque the haptic feedback system can apply (), we can determine this value with where and and are inputs to the function corresponding to the length of the user’s forearm and the orthogonal wrist joint force.

# **2.4 PWM**

The PWM will be used to control the motor and how much the coils are shorted. The team will use the timers provided by STMicroelectronics to generate the PWM and variably control its pulsations. The MCU will receive values from the host machine and will map those values to specific PWM values. The team will develop all drivers necessary for the PWM and corresponding timers. These drivers will initialize a GPIO to be controlled by a timer and initialize a timer to generate pulses at a specific frequency that last for a specific amount of time based on the rollover value and frequency prescaler.

**2.5 Transitioning device between sleep mode and active mode**

The transitioning between sleep and active mode will be handled via an analog event detector that generates external interrupts for the MCU to handle. These interrupts will be handled via a single wakeup and sleep handler which will be used to wake up/put to sleep other peripheral devices including the IMU and BLE chips via GPIO outputs. All timers, communication protocols, and pull up resistors will be disabled during deep sleep and re-initialized during the wake procedure.

3.0 Testing Plan

# **3.1 SPI (IMU) - 2**

The SPI and IMU will be tested concurrently since they both rely upon each other. The first test will verify communication via a digital logic analyzer/oscilloscope to watch the 4 serial lines and the who\_am\_i() function which issues a command to the IMU’s address and should receive the correct address back. According to the user manual, the address should be 0xEA.

The second test will continuously request to read the magnetometer, accelerometer, and gyroscope registers and print the formatted data out to the terminal via a UART connection. The test should see data change accordingly with the corresponding positional and rotational changes.

# **3.2 ADC/PWM (Battery and Analog potentiometer/PWM Output) - 3**

The ADC and PWM will be tested in tandem for general functionality and value mapping. The general functionality test will consist of a potentiometer that drives the ADC and then translates those values over to the PWM’s capture and compare value. The PWM should drive an LED such that varying the potentiometer’s resistance varies the duty cycle of the PWM and the brightness of the LED. Finally, the value mapping test will be similar to the general functionality except that it will use a table to translate the ADC’s value to the corresponding predefined PWM level.

The battery status functionality will be tested via a generated alert when the battery voltage level is deemed to be low. This alert could be in the form of a message on the host computer or LED. To test this function the team will continuously drain a battery until it reaches a specified reading on a voltmeter. Then the team will then plug the battery into the circuit and wait for the generated alert.

# **3.3 UART - 1**

The UART will be tested for general functionality of both the input and output lines. The team will test the UART communication via a digital logic analyzer/oscilloscope to help watch and decode the activity of the different lines. The test will consist of an echoback program that sends a byte from the MCU and echoes back the same byte from an Arduino over the UART connection. After the initial byte transmission the program should echoback whatever it receives on either end. After three cycles the byte should end on the Arduino and match the original byte.

# **3.4 Bluetooth Low Energy (BLE) Communication - 3**

Bluetooth communication will be tested for initialization and general functionality. The initialization of the BLE chip will be handled via UART commanding to configure the different register values. In order to test this initialization, two scripts of UART commands will be compromised to configure one chip as a central device, scan for other peripherals, and connect to a specific peripheral and then configure another chip as a peripheral device, advertise its existence, and bond to the central device. At the end of the script the two chips should be bonded and connected which can be seen using a status UART command. Additionally, to test the general functionality of communication, the team will use an echo test structure similar to the UART testing where the two endpoints are the two bluetooth chips being commanded by an Arduino and STM MCU. The MCU should initiate a transmission which is echoed back by the Arduino. Then the two devices should ping pong the original transmission data until the Arduino ends the test and prints out the final value. If the final value is the same as the initial value, then the test is passed.

# **3.5 Host Computer VR Simulation - 1**

All subsystems of the VR simulation will be tested visually in the VR scene. Namely, the primary elements that will be verified are the shoulder, elbow, and wrist joint locations as well as the force vector on the wrist joint.

# **3.5 Transitioning device between sleep mode and active mode - 4**

The transitioning between the sleep mode and active mode will be driven by a hardware analog event detector circuit which is separate from this scope. The event detector will generate interrupts which are handled by the MCU in order to make decisions within the state transition diagram. As such testing will require testing of the transition diagram as well ensuring that sleep mode deactivates any unnecessarily peripherals and devices and the active mode wakes all devices and re-initializes any necessary devices. To test this, LEDs will help indicate the various states and transitions (where “all-off” indicates sleep mode) and all the previous tests will be invoked upon transition from sleep mode to test the re-initialization of each peripheral/device.

4.0 Sources Cited:

[1] BleWinrtDll. 2021. Adam Brunnmeier. Accessed: October 6, 2021. Available: <https://github.com/adabru/BleWinrtDll>

[2] Fusion. 2019. x-io Technologies. Accessed: October 6, 2021. Available: <https://github.com/xioTechnologies/Fusion>

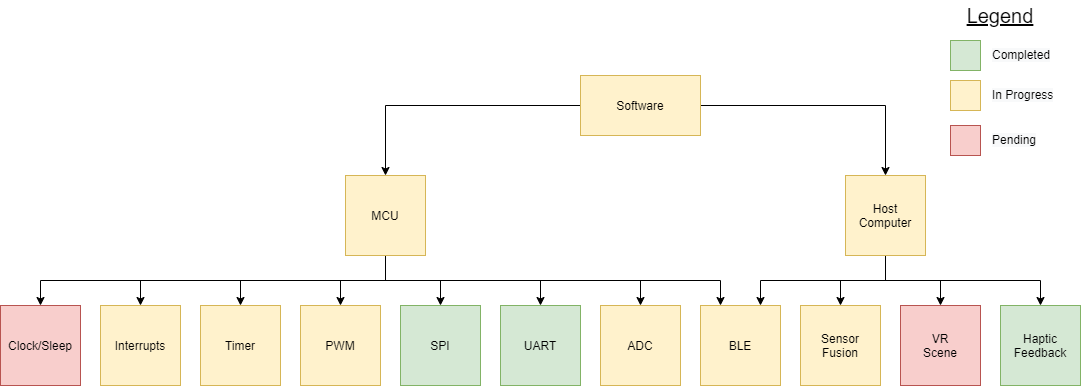
[3] Microchip Technology. 2014. RN4020 Bluetooth® Low Energy Module User’s Guide. Accessed: October 6, 2021. Available: <http://ww1.microchip.com/downloads/en/devicedoc/70005191b.pdf>

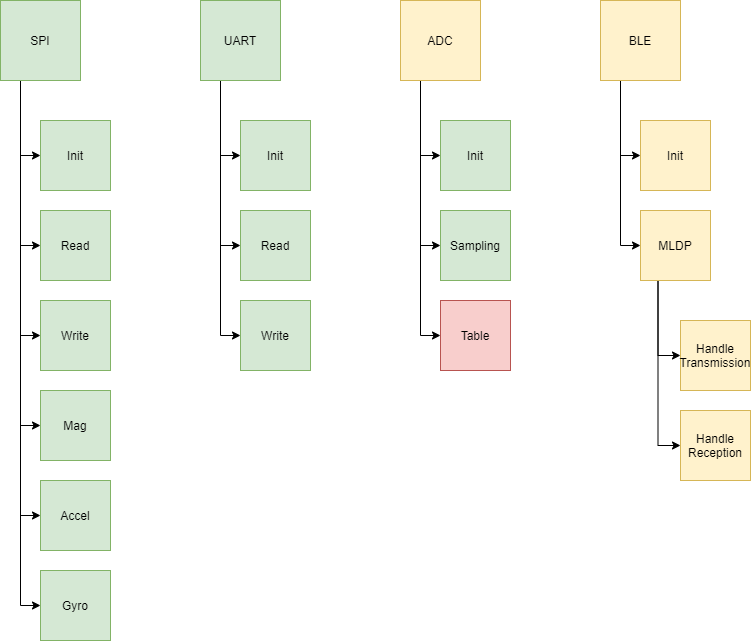
[4] Sparkfun. 2021. SparkFun ICM-20948 Arduino Library. Accessed: October 5, 2021. Available: <https://github.com/sparkfun/SparkFun_ICM-20948_ArduinoLibrary>

[5] STMicroelectronics. 2016. Description of STM32L0 HAL and Low Layer drivers. Accessed: October 6, 2021. Available: <https://www.st.com/resource/en/user_manual/dm00113898-description-of-stm32l0-hal-and-low-layer-drivers-stmicroelectronics.pdf>

[6] Unity. 2019. Unity Technologies. Accessed: October 6, 2021. Available: <https://unity.com>

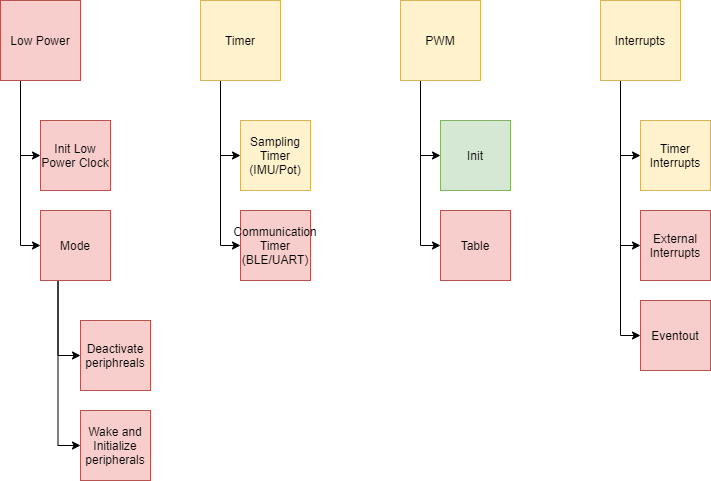
Appendix 1: Software Component Diagram



**

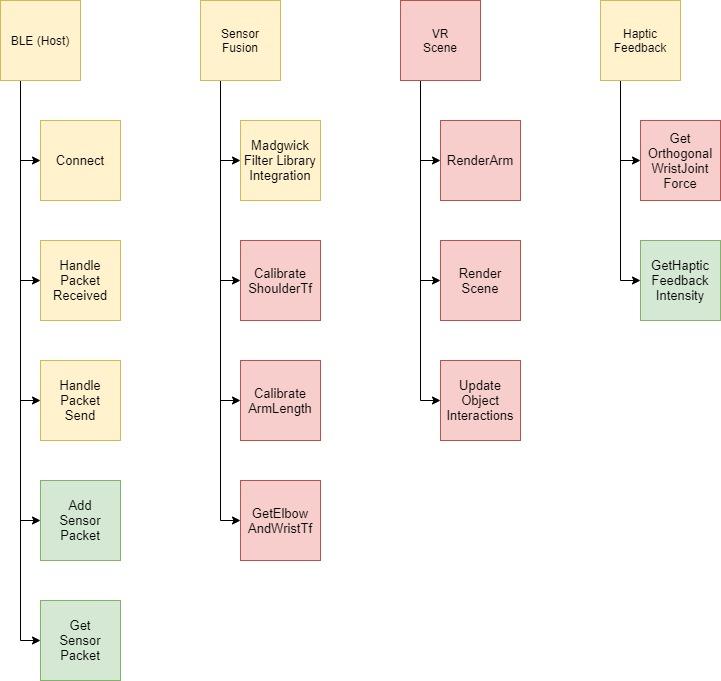
\*All Init blocks are assume to handle initialization of any required pinouts, timers, configurations, or preloaded values

\*Table refers to training and testing table lookup

**

\*All Init blocks are assume to handle initialization of any required pinouts, timers, configurations, or preloaded values

\*Table refers to training and testing table lookup

**