Software Overview

Year: 2021 Semester: Fall Team: #6 Project: RevEx

Creation Date: September 6, 2021 Last Modified: September 6, 2021

Author: Zach Ghera Email: zghera@purdue.edu

Assignment Evaluation:

| **Item** | **Score (0-5)** | **Weight** | **Points** | **Notes** |
| --- | --- | --- | --- | --- |
| **Assignment-Specific Items** | | | | |
| **Software Overview** |  | x2 |  |  |
| **Description of Algorithms** |  | x2 |  |  |
| **Description of Data Structures** |  | x2 |  |  |
| **Program Flowcharts** |  | x3 |  |  |
| **State Machine Diagrams** |  | x3 |  |  |
| **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 Software Overview

The following list breaks down the primary software functionality of RevEx:

* Read in sensor data
  + inertial measurement unit (IMU)
  + Analog potentiometer
* Bluetooth communication between microcontroller (MCU) and host computer
* VR simulation
  + Estimate arm position through sensor fusion
  + Enable users interaction with virtual objects
  + Calculate haptic feedback information
* Applying haptic feedback controls to motors
* Transitioning device between sleep mode and active mode

The implementation for each of these subsystems is detailed in the subsections below.

# 1.1 Read in sensor data

The RevEx wearable will contain two sensors to help estimate arm (wrist, elbow and shoulder joint) location: A 9 degree-of-freedom (DOF) IMU and an analog potentiometer.

# 1.1.1 IMU

The IMU will be positioned on the user’s upper arm (somewhere in between the elbow and the shoulder) in order to estimate the orientation of the upper arm relative to the user’s shoulder. The IMU contains a 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer. The MCU will read the raw 3D vectors from all 3 sensors over SPI. Multiple samples of the three IMU 3-D vectors will be included in the sensor packet as the IMU sampling rate will likely exceed the sensor packet send frequency (100 Hz) for greater accuracy in sensor fusion. See the Description of Algorithms section for more details on fusing all 3 IMU sensors to obtain upper arm orientation.

# 1.1.2 Analog potentiometer

The analog potentiometer will be placed on the elbow joint such that the resistance is changed when the user extends their forearm. The analog voltage read on the output pin of the potentiometer will be read into the MCU using the ADC. The ADC will likely have a higher sampling frequency than the packet send frequency (e.g. 500 Hz) to mitigate jitter (i.e. obtain a representative measurement). The hardware team will run experiments to create a mapping from the analog voltage output to the angle between the user’s upper and forearm (elbow angle). This mapping will be stored in a lookup table on the MCU’s Flash. The elbow angle from the average of the analog voltages since the last sensor packet was sent will be included in the next sensor packet.

# 1.2 Bluetooth communication between MCU and host computer

The RevEx wearable will contain a bluetooth low energy (BLE) module that is responsible for bi-directional communication between the microcontroller and the host computer. The MCU will communicate with the bluetooth module over SPI.

The bluetooth module should be treated as a black box with an input and output buffer that data will be written into and pulled from. Should transmission errors occur such as collisions, the microcontroller should retransmit the data after waiting an exponentially increasing random amount of time similar to how networks resolve collisions between hosts.

The different packet types transmitted between MCU and host are detailed in the Description of Data Structures section.

# 1.3 VR simulation

There are 3 major components to the VR simulation: estimate arm position, enabling users interaction with virtual objects, and calculating the haptic feedback information.

# 1.3.1 Estimate arm position

The arm position will be estimated by estimating the shoulder, elbow, and wrist joint locations.

At the start of the simulation, an initial calibration sequence will be used to determine shoulder location relative to the VR headset as well as the length of the user’s forearm and upper arm. This calibration sequence will ask the user to move their arms to various positions such that we can manipulate the known coordinate frames of the headset and controllers to infer other body locations (e.g. with trigonometry and basic euclidean distances). During this calibration sequence when the MCU is sending the host sensor packets, the host will throw the packets away and send back a “zero” haptic feedback packet (see Description of Data Structures for details) to ensure the connection with the MCU stays alive. Once the shoulder position is determined, the host computer will start to unpack new sensor data packets to determine the elbow and wrist joint locations.

As mentioned in section 1.1, sensor packets will contain multiple samples of 3D accelerometer vectors, 3D gyroscope vectors, and 3D magnetometer vectors. These vectors will serve as input to a Madgwick filter (Description of Algorithms for details) in order to determine the orientation vector of the upper arm relative to the shoulder. The absolute position of the shoulder, orientation of the arm, and the length of the upper arm from the calibration sequence will allow us to calculate the position of the elbow joint.

The other piece of information in the sensor packet will be a value indicating the elbow angle. The output of the Madgwick filter will allow us to determine the orthogonal vector that “goes through” the user’s bicep/tricep. Using that vector, the upper arm orientation vector, the forearm length, the elbow angle, and some simple trigonometry will allow us to calculate the position of the wrist joint.

# 1.3.2 Enable user interaction with virtual objects

Once the location of the user’s arm joints are determined, we can map these onto the user in VR. This way, when the users move their arms, their avatar in VR will move its arms to the same relative position. A simple scene will be constructed so the user can manipulate (e.g. move, pick up, or knockover) various objects in VR.

# 1.3.2 Calculate haptic feedback information

Once the user interacts with an object, the user’s current joint positions and details about the object will be fed into the subsystem for calculating the haptic feedback information. There are two components to the haptic feedback information packet:

1. Haptic feedback intensity: Indicates how much passive braking/resistance to apply to the elbow joint motor. Please see the Description of Algorithms section for details on this calculation.
2. Haptic feedback direction: Indicates whether to apply the haptic feedback for elbow extension, apply feedback for elbow flexion, or do not apply any feedback (in the case when haptic feedback intensity is 0).

# 1.4 Applying haptic feedback controls to motors

Haptic feedback will be induced through pulse width modulation (PWM) of a relay used to short the coils of a stepper motor.

Similar to the obtaining elbow angle from the analog potentiometer, there will be a lookup table that maps from haptic feedback value provided in the packet sent by the host computer into a PWM frequency. This output PWM signal will variably short the coils of our stepper motor to apply passive, haptic feedback. See the Theory of Operation section of the RevEx Functional Specification for more details on haptic feedback implementation.

The haptic feedback direction value will be used alongside new analog potentiometer voltage readings in order to ensure that haptic feedback is applied when the user extends or flexes their elbow in the desired direction.

# 1.5 Transitioning device between sleep mode and active mode

Transitions between sleep mode and active mode will be handled through an interrupt generated from the analog event detector circuit. If the device is in sleep and the detector generates an interrupt it should wake up and resume active mode.

Transitions from active mode to sleep mode will similarly use an interrupt generated by an analog event detector. If an interrupt is not generated after a certain number of cycles (e.g. using a countdown timer), then the device will enter sleep mode.

# 2.0 Description of Algorithms

# 2.1 Upper-arm orientation vector

The host-computer will utilize the Madgwick filter [1] to determine the orientation vector of the upper arm relative to the shoulder. We specifically plan to use the open source implementation of the Madwick filter in C# provided by x-io Technologies [2].

# 2.2 Haptic feedback “intensity”

The haptic feedback intensity value will indicate how much braking force we want to apply for haptic feedback. For example, this intensity value may be represented with 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 the length of the user’s forearm and the orthogonal wrist joint force that can be determined by the physics engine in the VR toolbox.

3.0 Description of Data Structures

As mentioned in section 1, the MCU will contain two lookup tables in flash. One that maps analog potentiometer voltage output to the angle between the user’s upper and forearm (elbow angle) and the other maps from haptic feedback value provided in the packet sent by the host computer into a PWM frequency. Depending on the relationship between ADC precision and power consumption (to be tested by the hardware team), the former lookup table will contain up to 8 bytes per table entry. The same goes for the latter lookup table.

The primary data structures of interest for RevEx are the different packet types that will be transmitted between the MCU and the host computer via BLE:

The first packet type is the sensor packet sent from the MCU to the host computer. This packet contains (where will likely be ~100 Hz and will be some integer multiple of ) sets of accelerometer, gyroscope, and magnetometer 3D vectors. Each vector element has 16-bit precision, meaning one IMU sample may be up to 18 bytes. In addition to the IMU data, the sensor packet includes the elbow angle. This will be stored as an integer that may have up to 32-bit precision.

The second packet type is the haptic feedback packet. This packet is sent from the host computer to the MCU to indicate haptic feedback intensity and direction. Haptic feedback intensity will be an integer that indicates how much passive braking/resistance to apply to the elbow joint motor. Depending on available resolution of the braking circuit, this integer may be a single byte or (if needed) up to 8 bytes. The haptic feedback direction value will only take on one of three values corresponding to the following braking states: apply the haptic feedback for elbow extension, apply feedback for elbow flexion, or do not apply any feedback (in the case when haptic feedback intensity is 0). Thus, this value will be represented as a 1 byte integer in the packet.

The final packet is the sleep packet sent from the MCU to the host computer. This packet will not contain any data as it is solely used for the MCU to inform the host that it is entering sleep mode and will try re-pairing over BLE on the next wakeup event.

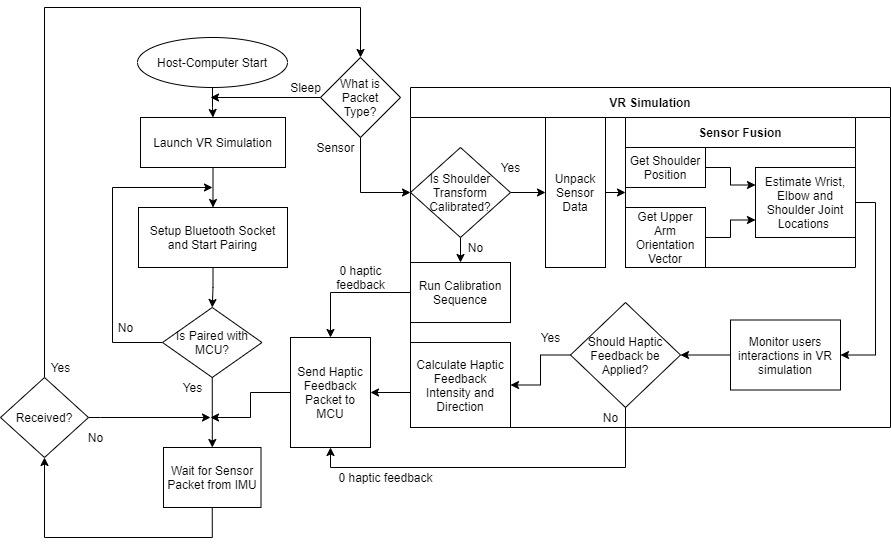
Apart from the packet structures listed above, the microcontroller and host computer will use primitive data structures for various use cases, such as a flag on MCU to indicate which state to enter next in the main loop, or an array representing the quaternion for the Madgwick filter.

4.0 Sources Cited:

[1] S. Madgwick, “An efficient orientation filter for inertial and inertial/magnetic sensor arrays,” *Report x-io and University of Bristol (UK)*, 2010, vol. 25, pp. 113–118. (<https://www.x-io.co.uk/res/doc/madgwick_internal_report.pdf>)

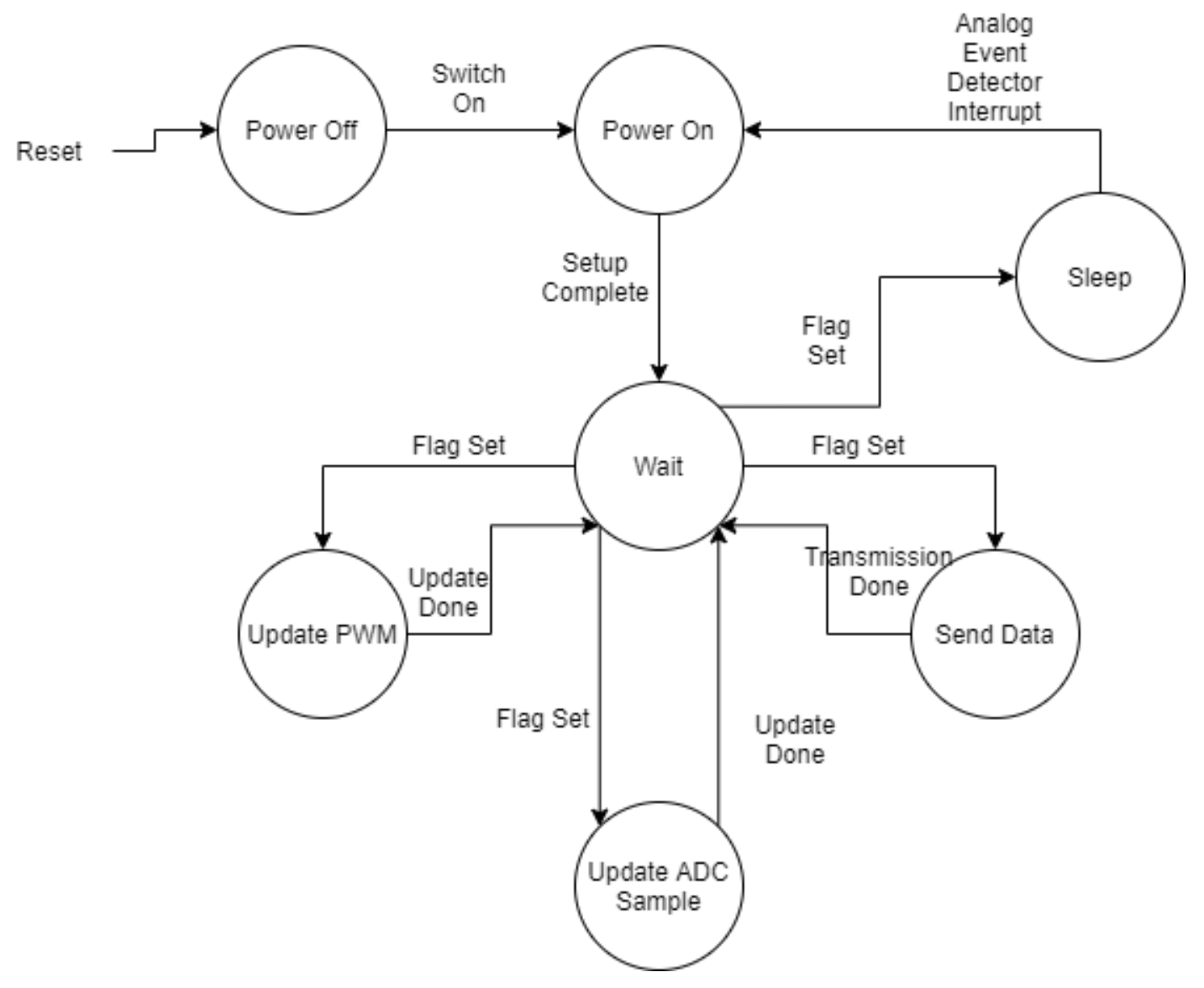
[2] x-io Technologies Limited, “Open source IMU and AHRS Algorithms,” *x-io Technologies*, <https://x-io.co.uk/open-source-imu-and-ahrs-algorithms/> (Accessed: 10-Sep-2021).

Appendix 1: Program Flowcharts

**

Host-Computer Flowchart

Appendix 2: State Machine Diagrams



Within the microcontroller there are three main states: power off, active, and sleep. The power off state is best described by, ideally, zero current flow through the system. The sleep state is best described by the absolute minimum current flow through the system. The active state is the most variable and interesting and can be flushed out into several other states as seen in the diagram above. While active the microcontroller will be using a timer and interrupts to constantly enter and exit out of a central wait state. This wait state will be exited using the assertion and deassertion of flags by the timer’s interrupt at a predetermined frequency for each branch. The priority of checking these flags can be observed as the highest priority on the left and lowest on the right of the diagram. Additionally, the wait state will always check for a flag set via an analog event detector. If after a certain number of interrupts an analog event is not detected, then the microcontroller will set a flag which moves the state machine into a sleep state until the next analog event.