Software Formalization

Year: 2022 Semester: Fall Team: 5 Project: Metaporter

Creation Date: October 1, 2022 Last Modified: October 1, 2022

Author: Jehan Shah Email: shah435@purdue.edu

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

For the most part, we are not using any third-party software on our microcontroller. The only part where we use external software on the microcontroller is to manage the LCD display. We have written our own code to transfer data to the display over SPI. However, we use certain utility functions to efficiently manage the display that were provided to us in ECE 362 and are originally adapted from examples by LCD Wiki. The adapted code performs the initialization sequence for the LCD, enables clearing the screen, and writing a character to specific area of the screen when it is put in the SPI\_DR register. This code is open-source and freely available.

We will be using many other third-party software packages on our Jetson Nano and host machine to process the raw sensor data and create the final reconstruction. Information about those is provided in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **License** | **Description** | **Use** |
| ROS 2 (Humble Hawksbill) | Apache 2.0 | Robot Operating System (ROS) is a software development kit for robotics applications. It provides drivers, algorithms, and developer tools. | We intend to use ROS as a middleware layer for communication. This will allow us to have a modular software architecture. For instance, we will have ROS nodes that stream camera and IMU data, and other nodes that subscribe to those and process them for downstream applications. |
| GStreamer | GNU Lesser General Public License | GStreamer is a library for multimedia handling. | We will be using a GStreamer pipeline for image capture from our camera. |
| OpenCV | Apache 2.0 | OpenCV is an open-source library that includes many computer vision algorithms. | We will use OpenCV for camera calibration, image manipulation and preprocessing. We may also optionally use OpenCV (core, calib3D) for our bonus implementation of a visual-inertial SLAM algorithm |
| Matlab (Computer Vision Toolbox, Sensor Fusion Toolbox) | Closed Source (Educational / Student License) | Matlab is a programming platform that provides highly performant toolboxes for various applications including computer vision. | Matlab provides functions for a simpler implementation of visual-inertial odometery than using its OpenCV counterparts and creating our own implementation. If we are not doing the bonus tasks, we intend to use these to compute camera poses. |
| Colmap | New BSD license | COLMAP is a general-purpose Structure-from-Motion (SfM) and Multi-View Stereo (MVS) pipeline with a graphical and command-line interface. It offers a wide range of features for reconstruction of ordered and unordered image collections. | We may use functions provided by Colmap as alternatives to functions in the Matlab Computer Vision toolbox to compute camera poses from images. Instant NGP (discussed below) provides a script that uses Colmap to get camera poses just from images. This will be used as a convenient early test. |
| Instant NGP | Nvidia Source Code License-NC  This requires that we (a) reproduce or distribute the work under the same license, (b) include a complete copy of the license, and © we retain all attributions present.  This is only provided for non-commercial use. For commercial use, a license can be requested. | The Instant Neural Graphics Primitives includes a hardware accelerated implementation of neural radiance fields (NeRF). | We will be using a NeRF as the final step in our pipeline to perform volumetric rendering of the target object. Given a set of RGB images and the camera poses the NeRF will give us a 3D model. |

2.0 Description of Software Components

For our software on the microcontroller, we are using an interrupt driven architecture. We have six main components: keypad, LCD, UART, IMU, timer, and DMA. For each of these we have a header file that defines the interface for that device and the corresponding C file with our function implementations. In general, each component has an initialization function that sets up the device for use, and one or two functions for the primary tasks performed by that peripheral.

Keypad: The user will interact with our device via the keypad. A button press on the keypad raises an EXTI interrupt. Within the interrupt service routine, we decode the keypress and set the device status accordingly. The interface for the keypad includes:

* void keypad\_init(void); This function initializes the relevant GPIO pins to their respective modes.
* int keypad\_get\_key (char\* returnChar); This function is called by the interrupt service routine and returns the key that way pressed. Based on the input, we modify our system state.

LCD: An LCD display gives the user important information about the state of the system. The interface for the LCD includes the following:

* void lcd\_setup (void); This initializes the display (using code adapted from LCD wiki) and the SPI subsystem on the microcontroller.
* void lcd\_update\_status (char\* status); This accepts a string and updates a particular part of the display with the status. Internally, this uses a function written by us to transfer data over SPI. Part of these functions were developed by members of the team in ECE 362. This also utilizes some code adapted from LCD wiki to physically update the display. This is called by the interrupt service routine that handles exti interrupts from the keypad.
* void lcd\_update\_timer (); Similar to lcd\_update\_status, this function updates a specific part of the screen to show the time elapsed since the user started recording data. This is called by the interrupt service routine for the timer.

IMU: An IMU is used to allow for better tracking and pose estimation for our device. We use a structure to programmatically represent the IMU. This struct allows us to keep track of important information such the device addresses, operating mode, the orientation data, and other useful information. Data is read from the IMU each time a timer generates an interrupt (described below). The interface for the IMU includes the following important functions:

* void imu\_init(IMU \* imu, uint8\_t addr, uint8\_t mode); This function intitializes the IMU by following the power on sequence and setting appropriate register. It also initializes the I2C subsystem on the microcontroller.
* void imu\_get\_quat(IMU \* imu); This function reads orientation data from the IMU in the form of quaternions. Internally, it utilizes I2C functions to send and receive data that were developed by me partly during ECE 362. This function is called by the interrupt service routine for timer 7.

There are many other utility functions that we have implemented to set various registers or modes on the IMU that are not shown here.

UART and DMA: We use UART to send data from the microcontroller to the Jetson Nano. Data sent includes commands to start and stop recording data, as well as data from the IMU. The interface for UART includes:

* void uart3\_init(void); This initializes the UART subsystem
* void uart3\_create\_header(uint8\_t\* pheader, uint8\_t command, uint8\_t d\_source, uint8\_t d\_type, uint8\_t num\_data); We use this to send a header preceding the data so that the Jetson Nano can interpret the data being sent. The figure below shows the tentative structure of our header.

A picture containing header diagram

Description automatically generated

* void uart3\_send\_byte(uint8\_t); Transmits a byte of data.

We use UART in conjunction with direct memory access (DMA) to transfer IMU data. The interface for direct memory access (DMA) includes:

* void dma1\_init(void); This initializes the DMA subsystem
* void dma1\_start(void \* src, uint32\_t dst, uint16\_t num\_bytes); This function transfers a given number of bytes from the source address to the destination address.

Each time a measurement is read from the IMU, we use direct memory access (DMA) to transfer data from the memory location containing the measurement to the UART data transfer register. Once data is put in the UART data transfer register, it is sent to the Jetson Nano. This allows us to continuously and efficiently transfer data from the microcontroller to the Jetson Nano. The dma1\_start function is called by the interrupt service routine for timer 7.

Timers: We are utilizing two timers to generate interrupts. The interface for each timer includes:

* void timX\_init(void); Initializes the timer to generate interrupts with a pre-defined frequency.
* void timX\_start(void); Starts the timer.
* void timX\_stop(void); Stops the timer.

We also write an interrupt service routine to handle the interrupt generated by each timer. Timer 7 is used to read data from the IMU at a rate of 100Hz. Timer 6 is used to update the display at a rate of 1Hz.

The software running on the Jetson Nano and the host machine is briefly described as it largely utilizes third-party software.

The Jetson Nano is expected to run multiple ROS nodes for the following tasks:

* Receive UART data
* Capture camera stream
* Multiple nodes to break down the SLAM process
* A node to publish camera poses

The host machine subscribes to the camera poses and camera images, and feeds those to the NeRF.

3.0 Testing Plan

1. I2C and SPI: This is the highest priority as debugging I2C and SPI can be quite difficult. We want to verify that our protocols are implemented correctly so that if we have issues during integration or with our final hardware, we are confident in the software. The software for I2C and SPI has been developed such that have a set of primitives for the protocol such as i2c\_send\_data or i2c\_recv\_data, and these are then called on by application-level functions. This allows us to test whether we have a mistake at the protocol or application level. We developed the primitives with support from materials from ECE 362 and the family reference manual. We tested these by completing a simple transaction with known devices. In the process, we step through the program and see that register values are being set correctly.

Once we had the primitives working, we started developing application-level functions incrementally. Each time we had to write to a register on a target device such as the IMU, we would view the transaction using the logic analyzer on the AD2 and verify that the correct values are being sent. This same approach was used with the LCD, and we then tested LCD application-level functions by also seeing if the LCD accurately rendered the information we were sending. This approach also allowed us to test for any timing issues.

2. UART: The second priority is testing UART. With UART working we can send sensor data to a host machine to visualize it. Therefore, it is an important tool to have working early. We developed code for UART using support materials from ECE 362. Testing UART is relatively simple. We send known strings to the host machine and the data received must match what was being sent.

3. Keypad: The keypad is crucial for user interaction; therefore, it is our next priority. To test the keypad, we need to test that we are setting the relevant GPIO pins correctly, we do this by using a voltmeter and debugging LEDs. Once we have that working, and we also have UART working, we read the key being pressed and send it over UART to verify.

4. DMA: DMA was easy to test with UART working. We created an array with a known string and passed the source address, destination address, and length to our function. We verified that this string was correctly sent over UART by viewing it on our host machine.

5. Timers: Timers are used to generate interrupts to call various subsystems. The most important thing that needs to be tested for the timers is that the prescaler and auto-reload values are set correctly as these dictate the frequency at which interrupts are generated. The most common mistakes here are if you are off by an order of magnitude, or off by one. We use a timer to send data over UART and see how long it takes us to send a certain amount of data. This helps us verify that we are in the correct order of magnitude. However, visual inspection of a high frequency timer has its limitations, so we also use the IDE debugger to check the values we set and verify the math for off by one errors.

6. IMU: Part of the IMU testing is already included in point 1. What we refer to here is testing whether the IMU is giving us meaningful and accurate values. To do this we will use visualization tools such as those provided by Matlab to see whether the orientation registered by the IMU accurately reflects it physically orientation. However, IMUs are prone to drift by nature so we will have to investigate what more we can do for testing.

7. High level software includes camera stream capture and visual-inertial slam. If we use third-party software, we may not have to perform much testing beyond ensuring that we are passing data in the correct formats. If we create our own SLAM implementation, we will verify it for correctness by benchmarking against known datasets with ground truth values.

4.0 Sources Cited:

[1] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, και W. Woodall, ‘Robot Operating System 2: Design, architecture, and uses in the wild’, *Science Robotics*, τ. 7, τχ. 66, σ. eabm6074, 2022. <http://docs.ros.org/en/humble/index.html>

[2] “News - gstreamer 1.20.3 stable bug fix release,” *GStreamer*. [Online]. Available: <https://gstreamer.freedesktop.org/>. [Accessed: 01-Oct-2022].

[3] G. Bradski, ‘The OpenCV Library’, *Dr. Dobb’s Journal of Software Tools*, 2000.

<https://opencv.org/>

[4] J. L. Schönberger και J.-M. Frahm, ‘Structure-from-Motion Revisited’, στο *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. <https://colmap.github.io/index.html>

[5] J. L. Schönberger, E. Zheng, M. Pollefeys, και J.-M. Frahm, ‘Pixelwise View Selection for Unstructured Multi-View Stereo’, στο *European Conference on Computer Vision (ECCV)*, 2016.

[6] T. Müller, A. Evans, C. Schied, και A. Keller, ‘Instant Neural Graphics Primitives with a Multiresolution Hash Encoding’, *ACM Trans. Graph.*, τ. 41, τχ. 4, σ. 102:1-102:15, Ιουλίου 2022. <https://github.com/NVlabs/instant-ngp>

[6] “Matlab Computer vision toolbox,” *Computer Vision Toolbox Documentation*. [Online]. Available: <https://www.mathworks.com/help/vision/index.html?s_tid=CRUX_lftnav>. [Accessed: 01-Oct-2022].

Appendix 1: Software Component Diagram





