Project: CS426 Spring 2018, Team #23: SkyWarden Senior Project, Aerial Drone Notification System (ADNS)
Team: Rony Calderon, Bryan Kline, Jia Li, Robert Watkins
Subsystem: Ground Base Unit
File name: Ground Base Unit Documentation.pdf
Description: Program documentation for Ground Base Unit subsystem

Program Documentation Overview:

Program Overview: High level description of the different modules and classes which make up the ground unit subsystem
Program Structure: High level description of how the different modules which make the ground unit subsystem work together and pass data between one another
Dependencies: Subsystem level description of the dependencies of the modules which make up the ground unit subsystem
Programming Units: Detailed description of each module and class, including the method used and their descriptions
Testing Modules: Description of the testing modules included in the system

Program Overview:

The program which reads the values coming in from the subsystem on-board the drone, controls the ground base unit hardware, interfaces with ROS, and which sends and receives data from the GUI subsystem is composed of several modules consisting of a number of classes and a main driver. The main driver creates all class objects needed to control GPIO pins associated with the LEDs, buttons, switches, and the speaker contained in the ground unit, as well as class objects to read in the values from the subsystem on-board the drone, send and receive data through ROS, and send data to and receive data from the GUI subsystem.

The main driver, `GroundUnitDriver`, makes use of the `GPIOZero` library to create and control `LED` and `Button` objects, and it creates `QuaternionManager, Parser`, `ROSNodeManager`, `GenericLEDs`, and `ShiftSevenSegment` and `PinSevenSegment` class objects in order to launch the quaternion XML file containing the sensor offsets, read in and parse data coming in from the drone, interface with the ROS master node, control the I2C bus controlling the generic LEDs, and to display the voltage value and proximity threshold on the seven segment displays, respectively. Additionally, the GUI subsystem is also run from the ground base unit as another process with which it exchanges data. Upon launching the system, the main driver creates all necessary objects, creates a number of processes and threads including those which control the GUI and speaker, it locates the serial port and connects with it, and in a main loop reads in the data from the drone over the serial port, parses the data, sends the data to the `ROSNodeManager` class and to the GUI subsystem, and controls the hardware in order to display the values and issue alerts to the drone operator.

In addition to the full system which reads in the values streaming in from the drone, displays the values and issues alerts on the ground unit hardware, publishes and subscribes to topics through ROS, and runs the GUI application, the software package also contains a lightweight, headless version of the system. The headless version of the system simply reads in the values from the serial port, parses them, and publishes them through ROS. This allows the drone operator to remove the RF transceiver from the ground unit which receives data from the drone, plug it into any other machine through USB, and stream and publish to ROS the data from that machine. In this way, the system can be customized for other purposes and it acts as a simple and efficient way to redirect the sensor data on-board the drone to the ROS master node.

Program Structure:

The main driver first instantiates a `Parser` class object which in turn instantiates a `SerialPort` object which searches for the serial port to which the RF transceiver is connected, sets the appropriate baud rate, and begins streaming in values from the drone. The `Parser` object further inspects and parses the data to ensure it is in the format the system expects. The system then creates a `ROSNodeManager` object and a `ShiftSevenSegment` object, and then a number of processes and threads are created to handle the hardware and other processes which the system runs. The first process that is created runs the GUI subsystem and a logical pipe is created in order to stream in the voltage value and stream out the voltage and proximity thresholds set in the GUI. Next, a process which runs the `ShiftSevenSegment` object's method to continually display a value to each display is established as is a pipe which sends the voltage value and proximity threshold into the process from the main loop. A thread is then created that continually monitors the lowest encountered proximity value and sets a variable to a constant corresponding to infinity periodically which refreshes the lowest value encountered which assists in determining whether or not the proximity threshold has been exceeded. Another process is created that checks whether or not an alert has been issued and if so activates the speaker. Next, a thread is created which controls the reset buttons on the ground unit and removes any bounce in them. Finally, a thread is created which creates a `QuaternionManager` object and invokes one of its methods to build the XML launch file from the configuration file and publish the sensor offsets as a static transform.

Once the necessary class objects, processes, pipes, and threads are created, the main driver enters the main loop wherein the `Parser` object is continually polled for data from the transceiver, the data is parsed and formatted and then passed on to the `ROSNodeManager` object to publish it through ROS. The data is also sent to the GUI and the GUI is polled for changes to thresholds made in the `SetThresholds` window. The state of the switches on the ground unit are packaged and passed into the `ROSNodeManager` object to publish their state, and the `ROSNodeManager` object also reports the state of the generic alerts, and the LEDs tied to the alerts are activated or deactivated accordingly by the `GenericLEDs` object. Lastly, the thresholds and reset buttons are checked and the voltage and proximity alert LEDs and the speaker are activated or deactivated accordingly. The main loop continues in this way until the drone operator kills the program.

Dependencies:

`GroundBaseUnit.py:`

time, threading, multiprocessing, gpiozero, subprocess, Parser, ROSNodeManager, GenericLEDs, quaternion_loader, ShiftSevenSegment, PinSevenSegment, ADNS_Main_GUI

quaternion_loader.py:

pathlib

Parser.py:

SerialPort

SerialPort.py:

serial, serial.tools, subprocess

ROSNodeManager.py:

geometry_msgs.msg, std_msgs.msg, rospy, visualization_msgs.msg, time

GenericLEDs.py:

subprocess, smbus, time

ShiftSevenSegment.py:

gpiozero, time

PinSevenSegment.py:

gpiozero, time

Programming Units:

GroundBaseUnit.py:

Class: None

Functions:

Name: getGenericSwitches

Description: Free function which takes in a list of Button objects, iterates through it and produces a list of flags holding the states of the Button objects which is used to send to the ROSNodeManager object in order to publish the states of the toggle switches

Parameters: Takes in a list of Button objects from gpiozero which correspond to the states of the toggle switches for generic alerts on the ground unit

Return: Returns a list of ints which hold flags for each of the states of the Button objects

Name: updateSevenSegment

Description: Free function used to establish a separate process which continually polls a Pipe object to receive voltage and proximity threshold data which are then sent to the SevenSegment object to display the values on the hardware on the ground unit

Parameters: Takes in a logical pipe which is used stream in values for voltage and the proximity threshold from the main loop into the process which updates the seven segments

Return: None

Name: resetProxMinCounter

Description: Free function which, running as a separate thread, declares a global value for the minimum proximity value encountered in a certain period of time, and then continually resets the minimum to a constant standing in for infinity so that a new minimum value can be established in order to refresh the minimum proximity value in the main loop which is used to issue an alert for proximity

Parameters: None

Return: None

Name: updateSpeaker

Description: Free function which creates a logical pipe between the main loop and the function running concurrently in a separate thread which sends in bools corresponding to whether an alert has been issue for voltage and proximity and the speaker is activated if either is true

Parameters: Takes in a Pipe object which sends in bools for the voltage and proximity alerts which

determine whether or not to activate the speaker
Return: None

Name: debounceButtons
Description: Free function which runs concurrently with the main loop in a separate thread which debounces the voltage and proximity alert reset buttons
Parameters: None
Return: None

Name: quaternionLauncher
Description: Free function which runs concurrently with the main loop in a separate thread which creates a roslaunch XML file from the sensor offsets configuration file and executes a system call to launch the XML to publish the quaternion values
Parameters: None
Return: None

quaternion_loader.py:

Class: QuaternionManager

Methods:

Name: __init__
Description: QuaternionManager class constructor which builds the quatList data member
Parameters: None
Return: None

Name: launchWriter
Description: QuaternionManager class method which creates an XML launch file from the data within the quatList 2D list containing sensor positional data. This method creates the launch file and overwrites any currently stored launch file. Each sensor is sequenced and all supplementary parameters vital for ROSmaster to use the sensor offset data such as package labels, method tags, and destination paths are also written into each node.
Parameters: None

Return: None

Name: loadValues

Description: QuaternionManager class method which takes in a the file name for stored quaternion data and loads it into quatList, the 2D class list. The file must be comma delimitted with 6 integers or floats per line (each line represents a sensor and each number represents x, y, z offsets and roll, pitch, yaw transforms respectively). The method tests that a file does exist and calls the launchWriter method after values have been loaded in.

Parameters: Takes in a string which represents the name of the file containing quaternion data.

Return: None

Parser.py:

Class: Parser

Methods:

Name: __init__

Description: Parser class parameterized constructor which takes in baud rate and timeout values which are used to create a SerialPort object

Parameters: Takes in two ints which are the baud rate and timeout value needed in the constructor of the SerialPort object

Return: None

Name: getSerialInput

Description: Parser class method which continually reads in from the serial port through the SerialPort object and formats and concatenates the portions of the input, removing artifacts added by PySerial and checking for a decimal point

Parameters: None

Return: Returns a list containing the value from the receiver as a string and char indicating what type of value it is

Name: trimValue

Description: Parser class method which takes in a list containing as the second element a string which is to have trailing digits trimmed off depending whether the string corresponds to a voltage or a proximity reading
Parameters: Takes in a list containing a control char marking the type of value which is being received from the serial port, which is the second element in the list
Return: Returns the list that was taken in but with the string containing the value trimmed down to the proper size for use in other classes in the ground unit

Name: parsePipeInput
Description: Parser class method which takes in a value and immediately returns it; provides for further parsing if necessary in the future by way of keeping other parsing methods intact
Parameters: Takes in a value which is forwarded
Return: Returns the value passed into the method

SerialPort.py:

Class: SerialPort

Methods:

Name: __init__
Description: SerialPort class parameterized constructor which takes in baud rate and timeout values which are used to create set up a Serial object
Parameters: Takes in two ints which are the baud rate and timeout value needed in the constructor of the Serial object
Return: None

Name: findSerialPorts
Description: SerialPort class method which iterates through all the serial ports looking for the ports in use preceded by either "ACM" or "USB" and returns that port as a string as the port number will often change and be assigned new values

Parameters: None

Return: Returns the name of the serial port in use as a string

Name: serialRead

Description: SerialPort class method which forwards the value intercepted from the Serial object on to the caller

Parameters: None

Return: Returns the value taken in from the serial port via the Serial object

Name: flushSerialPort

Description: SerialPort class method which invokes the Serial class method to flush the serial port buffer

Parameters: None

Return: None

ROSNodeManager.py:

Class: ROSNodeManager

Methods:

Name: __init__

Description: ROSNodeManager class default constructor which initializes the node and then creates a number of other nodes, both publishers and subscribers, by calling the various initializer methods

Parameters: None

Return: None

Name: initializeVoltageNode

Description: ROSNodeManager class method which initializes the voltage publisher node which continually publishes the voltage value

Parameters: None

Return: None

Name: initializeVoltageNode

Description: ROSNodeManager class method which initializes the proximity publisher node which continually publishes the proximity values

Parameters: None

Return: None

Name: initializeGenericSwitchNodes
Description: ROSNodeManager class method which initializes four generic publisher nodes which send signals corresponding to the state of toggle switches on the ground unit
Parameters: None
Return: None

Name: initializeGenericLEDNodes
Description: ROSNodeManager class method which initializes eight generic subscriber nodes which listen for ROS topics which cause LEDs on the ground unit to be activated if a generic alert is sent if the flags are high
Parameters: None
Return: None

Names: receiveLEDFlag_0 - receiveLEDFlag_7
Description: ROSNodeManager class call back methods which subscribe to topics from nodes called "genericLEDNode_0" to "genericLEDNode_7" and activates LEDs on the ground unit if the flags are high
Parameters: Take in an 8 bit int which acts as a flag to signal that a generic alert has been received
Return: None

Name: convertSensorDesc
Description: ROSNodeManager class method which takes in a sensor identifier as a char, converts it to an ordinal and appends it to the end of the quaternion label, which is returned as a string so that the correct sensor on-board the drone can be referenced for the quaternion launch file builder
Parameters: Takes in an identifier to convert to an ordinal
Return: Returns a string which is a quaternion label that can be used when relating the current proximity reading with the corresponding sensor on-board the drone

Name: publishSensorValue

Description: ROSNodeManager class method which takes in a value, checks whether it is a voltage or a proximity reading, and publishes the value either as a voltage topic or a proximity topic, if it is a proximity reading then it is first converted into a marker message which is tied to the quaternion of the corresponding sensor on-board the drone
Parameters: Takes in a sensor value, either a voltage or a proximity, and publishes it to the appropriate topic
Return: None

Name: publishGenericSwitches
Description: ROSNodeManager class method which iterates through the list containing the states of toggle switches on the ground unit and publishes each of their states
Parameters: Takes in a list of values corresponding to the state of the toggle switches on the ground unit
Return: None

Name: publishSwitchValue
Description: ROSNodeManager class method which publishes the state of a toggle switches on the ground unit at a particular position in the list that holds their states
Parameters: Takes in an int which is the index in the generic switch list and a value then publishes that value to the generic alert at the index specified
Return: None

Name: subscribeGenericLEDs
Description: ROSNodeManager class method which creates a list of ints, iterates through the flags tied to generic alerts which are being subscribed to, and sets the ints to the states of the alerts flags, and returns the list, which is used to determine which LEDs on the ground unit to activate
Parameters: None
Return: Returns a list of flags corresponding to the generic alerts tied to LEDs on the ground unit

Names: subscribeLEDValue
Description: ROSNodeManager class method which initializes eight generic subscriber nodes which listen for ROS topics which cause LEDs on the ground unit to be activated if a generic alert is sent if the flags are high
Parameters: Takes in an int which is the index in a list and checks the generic LED related to the the flag of that index value, and then returns the value of the flag
Return: None

GenericLEDs.py:

Class: GenericLEDs

Methods:

Name: __init__
Description: GenericLEDs class default constructor which initializes the MP121 I2C bus which drives the LEDs for the eight generic alerts on the ground base unit
Parameters: None
Return: None

Name: clear
Description: GenericLEDs class method writes the hexadecimal value 0x00 to the I2C bus which sets all pins to low, thereby turning off all generic LEDs on the ground unit
Parameters: None
Return: None

Name: setLEDs
Description: GenericLEDs class method which takes in a hexadecimal value which is built outside of the class and which codes the state of each of the eight generic alerts, the value is written to the bus, which in turns drives the eight LEDs which act as the generic alerts on the ground unit
Parameters: Takes in a hexadecimal value which corresponds to the eight generic alerts where a one is high and a zero is low
Return: None

ShiftSevenSegment.py:

Class: ShiftSevenSegment

Methods:

Name: __init__
Description: ShiftSevenSegment class default constructor which initializes the LED objects for the clock, enable, data, and digit select inputs to off
Parameters: None
Return: None

Name: selectDigit
Description: ShiftSevenSegment class method which takes in an int which is the the digit on the display to write, writing that digit low, thereby turning it on, for both the voltage and the proximity threshold at the same time
Parameters: Takes in an int which corresponds to the digit on the seven segment being written to, as each digit must be written individually
Return: None

Name: hexadecimalConversion
Description: ShiftSevenSegment class method which takes a number, the numbers to write to a given digit on the seven segment display and then builds the hexadecimal value corresponding to the segments to drive high, also taking in a bool corresponding to whether or not the decimal point should be written and if so the hexadecimal value is altered to reflect that voltage value
Parameters: Takes in a char corresponding to the number to write to the current digit on display, and a bool which signals whether or not the decimal point will be written
Return: Returns the hexadecimal value which was built from the char and the bool input into the method, representing the number to display

Name: shiftIn

Description: ShiftSevenSegment class method which takes in two hexadecimal numbers for voltage and proximity threshold and shifts them into the two shift registers for both displays simultaneously
Parameters: Takes in two hexadecimal numbers which are digits on the voltage and proximity threshold displays which will be shifted into the shift register
Return: None

Name: displayNumber
Description: ShiftSevenSegment class method which takes in two strings to write the seven segment displays, and moves through each of them and sends each char in the strings into another class method which builds hexadecimal values from them and then shifts those values into the shift registers for each display, and then latches that into two D-latches which feed into the displays
Parameters: Takes in two strings corresponding to the entire voltage and proximity threshold values to be written to the seven segment displays
Return: None

PinSevenSegment.py:

Class: PinSevenSegment

Methods:

Name: __init__
Description: PinSevenSegment class default constructor which initializes the display by zeroing out the pins and digits
Parameters: None
Return: None

Name: allPinsOff
Description: PinSevenSegment class method which writes all digit pins high which turns them off
Parameters: None
Return: None

```
Name:          selectDigit
Description:   PinSevenSegment class method which takes in
               the digit to be activated, turns all digits
               off, and then activates the the one passed
               in as a parameter
Parameters:    Takes in an int corresponding to the digit
               on the display which will be activated
Return:        None

Name:          allOn
Description:   PinSevenSegment class method which turns all
               digits and all segments on for
               testing purposes
Parameters:    None
Return:        None

Name:          allPinsOff
Description:   PinSevenSegment class method which takes in
               strings corresponding to the voltage value
               to be displayed on the seven segment and the
               proximity threshold which is only included
               so as to make the interface compatible with
               the class method's use elsewhere in the
               program and which is discarded
Parameters:    Takes in two strings, the voltage value and
               the proximity value, both as strings
Return:        None

Name:          numberSelect
Description:   PinSevenSegment class method takes in a char
               which is the number to be written to the
               digit and a bool corresponding to whether or
               not the decimal point will be written, and
               inspects the number to determine which
               segments to write high
Parameters:    Takes in the number to write to the digit as
               a char and a bool which determines whether
               or not the decimal point will be written
Return:        None
```

Testing Modules:

In addition to the full system and the lightweight, headless version of the system which implements only the connection from the drone transceiver to the ROS master node, the system also includes several modules which are necessary for testing the operation of the subsystem with regard to how it interfaces with ROS. The main way in which this is accomplished is by developing a series of modules which act as different parts of the ROS master node, including subscribing to the the drone voltage and proximity data, subscribing to the generic switches on the ground base unit, and publishing the generic alert flags to which the ground unit subscribes. To test that the system is properly publishing the voltage from the drone, the `voltagePublisherTest` script can be run and to test that the proximity is being published the `proximityPublisherTest` script can be run. Two different scripts, `genericLEDPublisherTest` and `parameterizedgenericLEDs`, both test the generic LEDs on the ground unit by publishing values on eight different ROS nodes, the first with hard coded values and the second which takes in values from the command line. The `genericSwitchSubscriberTest` script subscribes to the ROS topics tied to the switches on the ground unit and continually prints their values to the terminal. In addition to these scripts a number of other scripts test the quaternion launcher, and an earlier iteration of the static transform which the quaternion launcher replaced, as well as a basic scaffold around which the tests were built which act as a means of testing the test modules. The test modules are not necessary for the functioning of the system, however they are included here as they are useful tools to ensure the ROS interfacing module is correctly handling the data from the drone.