Unmanned Underwater Vehicle Software Documentation

Olympic College Engineering Club

NOTE: *This is a "**living document**", and as such will be updated on a semi-regular basis. Please ensure you have the **latest version** of this document at all times.*

Table of Contents

Overview

This document serves as a comprehensive reference for all software programs necessary for the operation of Olympic College Engineering Club's (OCEC) Unmanned Underwater Vehicle (UUV). This is a "living document", and as such will be updated on a sem-regular basis. Please ensure you have the most recent version of this document at all times. This document consists only of a list and explanation for the software of the project, and does not contain any source code or compiled binary code. Please see the appropriate section on the project's GitHub repository for the source code.

Software tasks on board the UUV are handled by two main components: A miniature computer called a Raspberry Pi (RasPi), and a microcontroller called an Arduino. The Raspberry Pi is the more powerful of the two, so the majority of computational tasks are run there. Programs for the RasPi are written primarily in Python 3.0, which is considered a beginner-friendly programming language. The Arduino is responsible for controlling servos and motors by sending the control signals that those components expect, as well as receive data from sensors and reporting it to the Raspberry Pi. Programs for the Arduino are written in C++, which is a somewhat more involved language that may be challenging for beginners to learn. Arduinos do not have and operating system to handle multiple programs running at once, so only a single program can be uploaded to and run by the Arduino at any one time. To make the software easier to write, maintain, and understand, the Arduino program is broken down into smaller parts that the "main" program will compile into a single binary file, which gets uploaded to the Arduino.

In addition to the UUV's onboard computers, a laptop computer is used by the operator to control the vehicle and view telemetry coming from it. The program for interfacing with the UUV from this "operator computer" is written in Python, and the operator computer will communicate with the UUV's Raspberry Pi over an Ethernet connection. The operator computer will also be connected to a gamepad controller, which is the primary means of controlling the vehicle. Controller input data are read by the operator computer, and sent to the vehicle's onboard RasPi for processing. Video data from the vehicle's camera and sensor data from the various sensors are displayed on the operator computer screen as part of the operator program.

The RasPi and Arduino communicate over a serial connection between them. Arduino has built-in functionality to send and receive data over serial, and the Raspberry Pi software will use the Python serial library to read and send data. Messages between the two are in a standardized form to make communication predictable and modular.

Arduino

Overview

The UUV uses an Arduino Mega 2560 for most of the interaction between the software layer and the electronics layer. The digital and analog pins on the Arduino connect to sensors (eg. thermistors, gyroscope, accelerometer, etc), thruster motors via electronic speed controllers (ESCs), servos, and any other peripheral devices on board the vehicle. The Arduino communicates with the UUV's onboard computer (the Raspberry Pi) to report sensor data and receive commands for motor power and servo angle, etc. Programs for the Arduino are written in C++, and the Arduino can only have one program uploaded & running at any time. To make programming for the Arduino easier to manage and more collaborative, its software is divided into smaller sub-programs, each of which contain the functionality for a single aspect of the Arduino's responsibilities, and which is combined into the "main" program at compile time using the "include" keyword in C++.

List of Tasks

The following is a list (in no particular order) of all the tasks that the Arduino program needs to be able to complete. Each item on this list can be partitioned into its own sub-program.

- Send and receive serial data to and from the Raspberry Pi.
- Interpret received commands from the RasPi and call corresponding functions.
- Package data from sensors into messages to send to RasPi.
- Set motors (via ESCs) to a specific speed value.
- Set servo(s) to a specific angle value.
- Read sensor data from analog sensors and digital sensors. Each sensor will have its own sub-program that interfaces with the sensor according to the datasheet and pinout of the sensor.

Each of these tasks is elaborated on below.

Main

The main program of the Arduino is the program that is used when compiling and uploading to the Arduino. In order to use functions from the sub-programs, each sub-program should be included in the main program using the "include" keyword at the beginning of the file. This will also require the sub-program's file to be located in a directory in the "include path" of the compiler (or the include path of the compiler is updated to add the directory where the sub-programs are already located). The main program is also where global variables are declared & defined, so that they can be passed to any functions that need them.

Arduino programs are made up of two primary functions: the "setup" function, and the "loop" function. The setup function runs once whenever the program starts, which happens when the program is uploaded to the board, when the board is powered up if the program is already uploaded, and also if the "reset"

button on the board is pressed while the board is running. The “setup” function in the main program should be used for tasks like setting the serial baud rate (the rate that signals are sent over the serial connection), initializing any objects that are used in the program (eg. “Servo” objects for servos and ESCs), etc. The loop function runs after the setup function completes, and it also runs every time it finishes, which results in it running indefinitely as long as the Arduino is powered. This is where the majority of calls to the functions from sub-programs are located, as many of them need to be constantly operating for as long as the vehicle is powered on. Examples of tasks that need to be completed every loop cycle are: reading the information from the serial connection to the Raspberry Pi, reading the values of all connected sensors, updating speeds and angles for ESCs and servos, sending telemetry to the Raspberry Pi over the serial connection, etc. Keeping the loop function readable is very important, which is part of why the tasks of the Arduino are partitioned off into sub-programs. Descriptive function names that only take one line of code are easier to read than long sections of loops and conditional statements that achieve that same thing.

Motor Control

The motors in the thrusters that move the UUV are controlled by the Arduino. The thruster motors are a type of motor called brushless DC motors, which are not as straightforward to control as traditional brushed DC motors. To use brushless motors, a type of microcontroller called an electronic speed controller (ESC) must be used. ESCs take a type of signal called a pulse-width modulation (PWM) signal which encodes the desired speed of the motor, and use it to activate the coils in the motor in such a way that it spins at that speed. The Arduino can output a PWM signal from certain digital pins on the board. The motor control program makes use of an Arduino library for controlling servos which can also be used to control ESCs, since servo motors and ESCs are both controlled using PWM signals.

Pulse-Width Modulation (PWM)

Pulse-width modulation is a technique for encoding data in a digital signal. PWM works by turning the output on and off 500 times per second, and encodes information by varying how long each cycle is turned on. The duration of each cycle is called the “pulse width” or “duty cycle”, and is measured in microseconds. For example, a pulse width of 2,000 microseconds would mean the output is always on, since 2,000 microseconds is the duration of each cycle when the frequency is 500Hz. A pulse width of 0 microseconds would mean the output is always off. A pulse width of 1,000 microseconds would mean the output is on for half the duration, and off for the other half.

Electronic Speed Controller (ESC)

An electronic speed controller is a type of microcontroller that is used to control brushless DC motors. Brushless motors, unlike simple brushed DC motors, have three sets of coils that need to be activated in a specific sequence in order to make them spin. The role of the ESC is to take as an input the desired speed of the motor (encoded as a PWM signal) and energize & deenergize the coils of the motor to cause it to spin at the desired speed. The valid PWM input range for the ESCs in this project is from 1,000 microseconds (“minimum”/lowest throttle) to 2,000 microseconds (“maximum”/highest throttle).

ESCs generally have a total of eight wires. The two largest wires are the power & ground connection that connect to a power source and provide the power needed to spin the attached motor. On the same side as these wires are a cluster of 3 smaller wires. The white wire in this cluster is the PWM input wire, which connects to the Arduino. The other two wires in the cluster can be used to supply 5 volt power from the ESC's power & ground connectors to the Arduino, but they are unused in this project. The three wires that come out the other side of the ESC, which are slightly smaller than the main power & ground connection, connect to the motor's three wires to power the motor. The order that the ESC's output wires connect to the motor's input pins doesn't matter, but swapping any two wires will reverse the "forward" direction of the motor when powered.

If necessary, an ESC can be reprogrammed to configure it to a specific use case. For the UUV's thrusters, the ESCs should be set to run in bi-directional mode. In this mode, the "minimum" throttle value corresponds to full reverse throttle, and the "maximum" throttle value corresponds to full forward throttle. The zero point in bi-directional mode is at halfway between "minimum" and "maximum" throttle, or a 1,500 microsecond pulse width.

Serial Communication

The Arduino communicates with the Raspberry Pi using a serial communication protocol called UART over a USB connection. ...

Baud Rate

The "baud rate" of serial communication refers to the number of bits sent across the connection per second. Higher baud rate means faster communication, but also can lead to more frequent errors in the data transmission. In order for devices to communicate with each other, the baud rates each of them is configured to use must be the same. If the baud rates are not the same, the data received will be unusable, if it's received at all. A common (but relatively slow) baud rate for Arduino communication is 9,600 baud, or 9,600 bits per second. Faster baud rates are generally multiples of this number, such as 57,600 baud or 115,200 baud. The communication between the Arduino and Raspberry Pi in this project uses 115,200 baud. ...

UART

Universal Asynchronous Receiver-Transmitter (UART) is a serial communication protocol which is used by Arduino to communicate over USB. ...

Data Buffer

...

Reading Data

...

Writing Data

...

Raspberry Pi

Overview

The Raspberry Pi is the main computer onboard the UUV. It is responsible for receiving commands either from the laptop computer of the operator on the surface if operating tethered, or creating commands (possibly? Depends on how we decide to make this work) onboard. These commands come mainly in the form of directional input from a controller, so the RasPi computes what power level each motor should be at to achieve the desired motion. Motor commands are then sent to the Arduino, which sends the signals to the motors. The RasPi also reads information from the serial connection to the Arduino, mainly sensor data from the vehicle's sensors. The sensor data is used to adjust the control of the vehicle, for example using gyroscope information to correct for unexpected roll, etc. Sensor data will also be sent to the laptop of the operator to be displayed. Connected to the RasPi is the UUV's camera, and the video feed from the camera is sent to the operator computer to give the operator a first-person view of the vehicle. When running autonomously, the RasPi will also be responsible for running the computer vision program that recognizes objects in the field of view of the camera.

List of Tasks

The following is a list (in no particular order) of all tasks the Raspberry Pi needs to be able to complete. Each item on this list can be its own separate Python program, but it needs to be included in the "main" program's "import" list at the start in order to be used.

- Receive commands from surface operator's laptop computer.
- Interpret and process commands from operator computer and send appropriate instructions to Arduino.
- Send commands to Arduino over serial connection.
- Receive sensor data from Arduino over serial connection.
- Process sensor data, and use it to inform the desired system state of the vehicle when calculating motor speed values, etc.
- Read camera feed from onboard camera and stream the feed to operator computer.
- Send sensor data to operator computer.
- (When in Autonomous Mode) Analyze environment data from sensors & camera to create virtual map of surroundings.
- (When in Autonomous Mode) Use virtual map for object avoidance & mission objective completion.

Each of these tasks is elaborated on below.

Main

The main program running on the Raspberry Pi is where calls to all the other functions are, along with global variables and the "import" keywords for all the other python scripts that the RasPi will need

access to. In order for the “import” keyword to find the script, it needs to be located in a directory listed in the “import path” of the Python environment. Unlike Arduino, the Python language does not have built-in “setup” and “loop” functions. Any tasks that need to be run once, on startup, can be either at the beginning of the script, or in a custom function which is called near the start of the script. Tasks that need to repeat indefinitely while the program is running should be placed in a loop that runs forever, or in a function which itself is in a loop that runs forever. This will achieve the same effect as the “setup” and “loop” functions found in Arduino programs. ...

Operator Computer

Overview

The operator computer is the human interface with the UUV. It can be any personal computer with an Ethernet port and a USB port, but is assumed to be a laptop computer. The UUV and the operator computer communicate over an Ethernet cable (which is part of the vehicle's tether system) using statically configured IPv4 addresses. Connected to the operator computer via USB is the gamepad controller which serves as the primary method for controlling the vehicle. The program which runs on the operator computer reads input data from the gamepad and sends the data over the Ethernet cable to the UUV's Raspberry Pi. Data from the RasPi, such as video feed from the camera and sensor readings, are received by the operator computer, and the operator program displays them for the operator to see. The operator program may also be used to send commands to the vehicle, either for testing & debugging purposes, or to allow specific control over any of the vehicle's functions. ...