(12) UK Patent Application (19) GB (11) 2551490 (13) A

(43) Date of A Publication 27.12.2017

(21) Application No: 1610416.8

(22) Date of Filing: 15.06.2016

(71) Applicant(s):
**Saab Seaeye Limited**
**(Incorporated in the United Kingdom)**
**20 Brunel Way, Segensworth, Farnham, Hampshire, PO15 5SD, United Kingdom**

(72) Inventor(s):
**Andrew Crosher**
**Carl Pettit**
**Mark Pettit**

(74) Agent and/or Address for Service:
**Reddie & Grose LLP**
**The White Chapel Building,**
**10 Whitechapel High Street, London, E1 8QS,**
**United Kingdom**

(51) INT CL:
***G06F 9/445*** (2006.01) *B63G 8/00* (2006.01)

(56) Documents Cited:
**WO 2013/126058 A1** **WO 2011/075139 A1**
**US 7480907 B1** **US 20050160257 A1**
**US 20040034861 A1**

(58) Field of Search:
INT CL **G06F**
Other: **WPI, EPODOC, TXTA, Internet**

(54) Title of the Invention: **Method and apparatus for updating software on remotely operated vehicle**
Abstract Title: **Updating Computer Executable Instructions in an Underwater Remotely Operated Vehicle (ROV)**

(57) A method of updating computer executable instructions (e.g. software or firmware) for one or more functional modules in an underwater remotely operated vehicle (ROV) comprises: receiving updated instructions at a functional module of the ROV from a surface control computer over a communication link; storing the updated computer executable instructions in a first portion of a non-volatile memory that is accessible by the processor of the functional module; and subsequently writing the updated computer executable instructions from the first portion of the non-volatile memory to a second portion of a non-volatile memory wherein the functional module is configured such that the processor executes computer executable instructions stored in the second portion of non-volatile memory. Determining whether updated instructions are stored in the first portion comprises executing predetermined instructions according to an "action" code stored in a portion of non-volatile memory accessible by the processor which indicates whether updated instructions are stored; if successful, this causes the updated instructions to be written to the second portion of non-volatile memory; or, if instructions are not stored, the action code causes the processor to execute pre-existing code in the second memory portion.

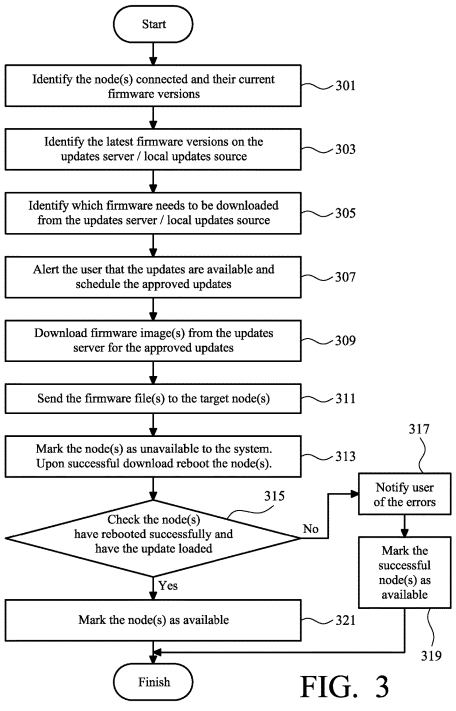


FIG. 3

GB 2551490 A

At least one drawing originally filed was informal and the print reproduced here is taken from a later filed formal copy.

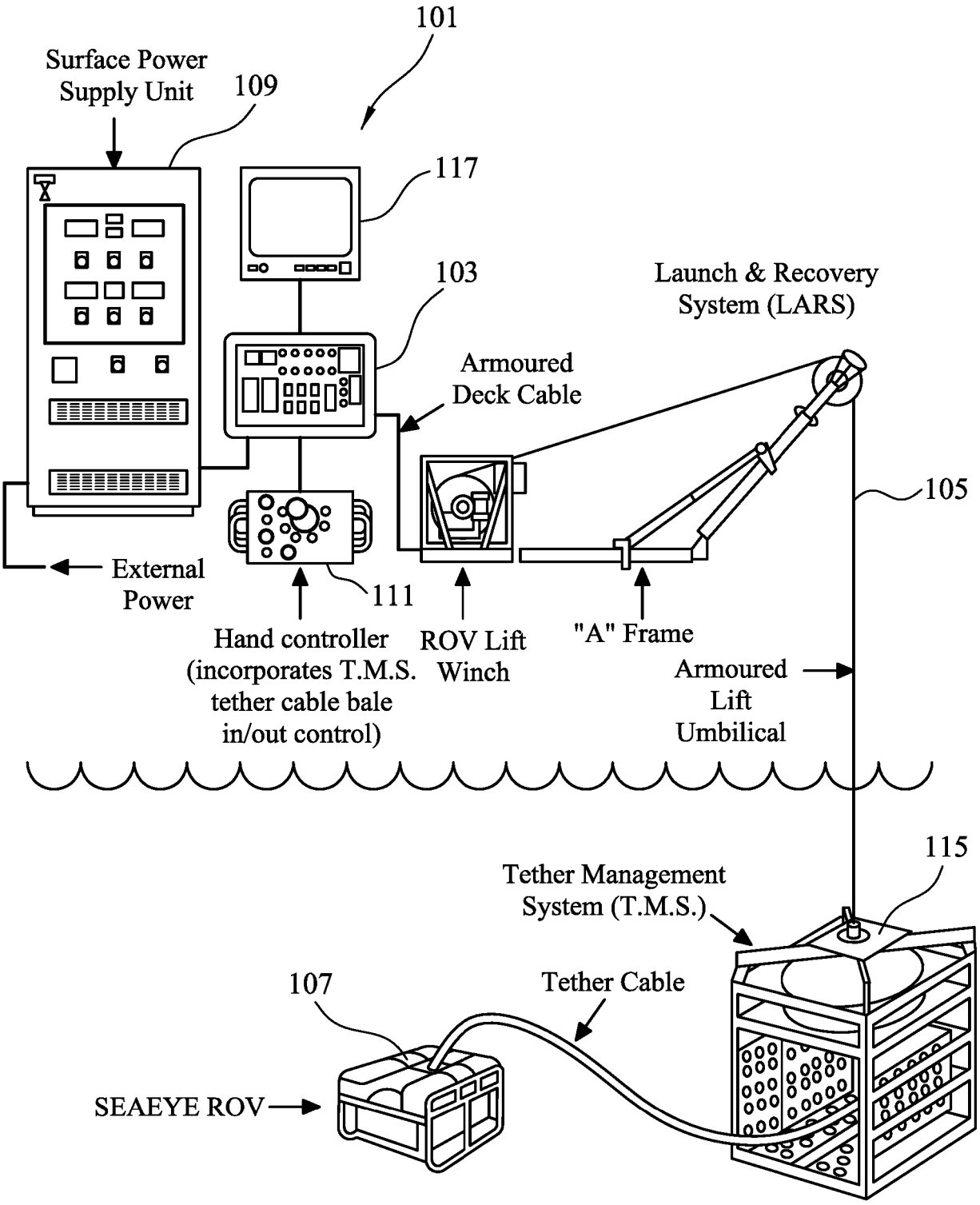
101
Surface Power Supply Unit
109
117
103
Launch & Recovery System (LARS)
Armoured Deck Cable
105
External Power
111
Hand controller (incorporates T.M.S. tether cable bale in/out control)
ROV Lift Winch
"A" Frame
Armoured Lift Umbilical
115
Tether Management System (T.M.S.)
107
Tether Cable
SEAEYE ROV

FIG. 1

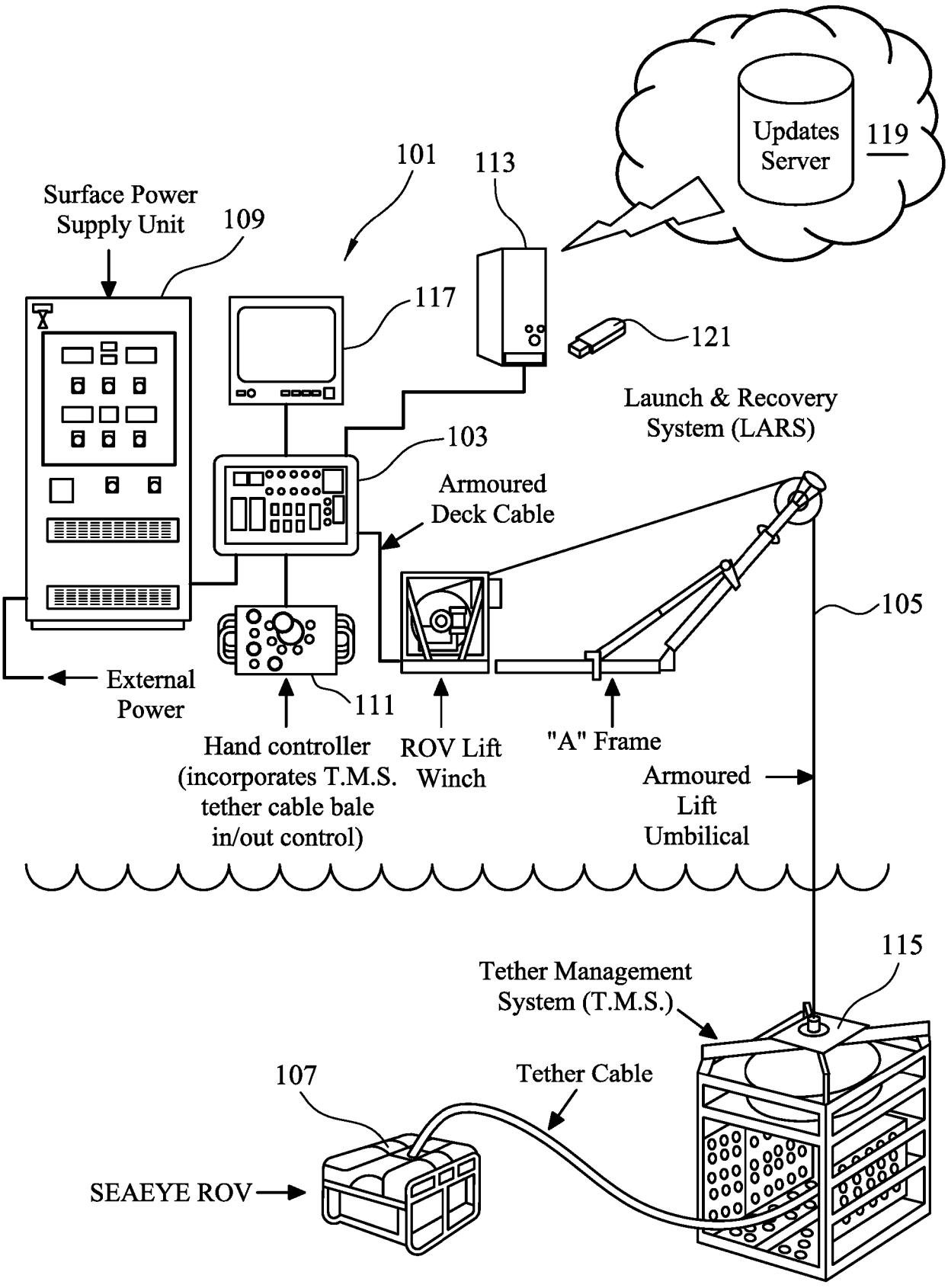
Updates Server
119
101
113
Surface Power Supply Unit
109
117
121
Launch & Recovery System (LARS)
103
Armoured Deck Cable
105
External Power
111
Hand controller (incorporates T.M.S. tether cable bale in/out control)
ROV Lift Winch
"A" Frame
Armoured Lift Umbilical
115
Tether Management System (T.M.S.)
107
Tether Cable
SEAEYE ROV

FIG. 2

14 09 17

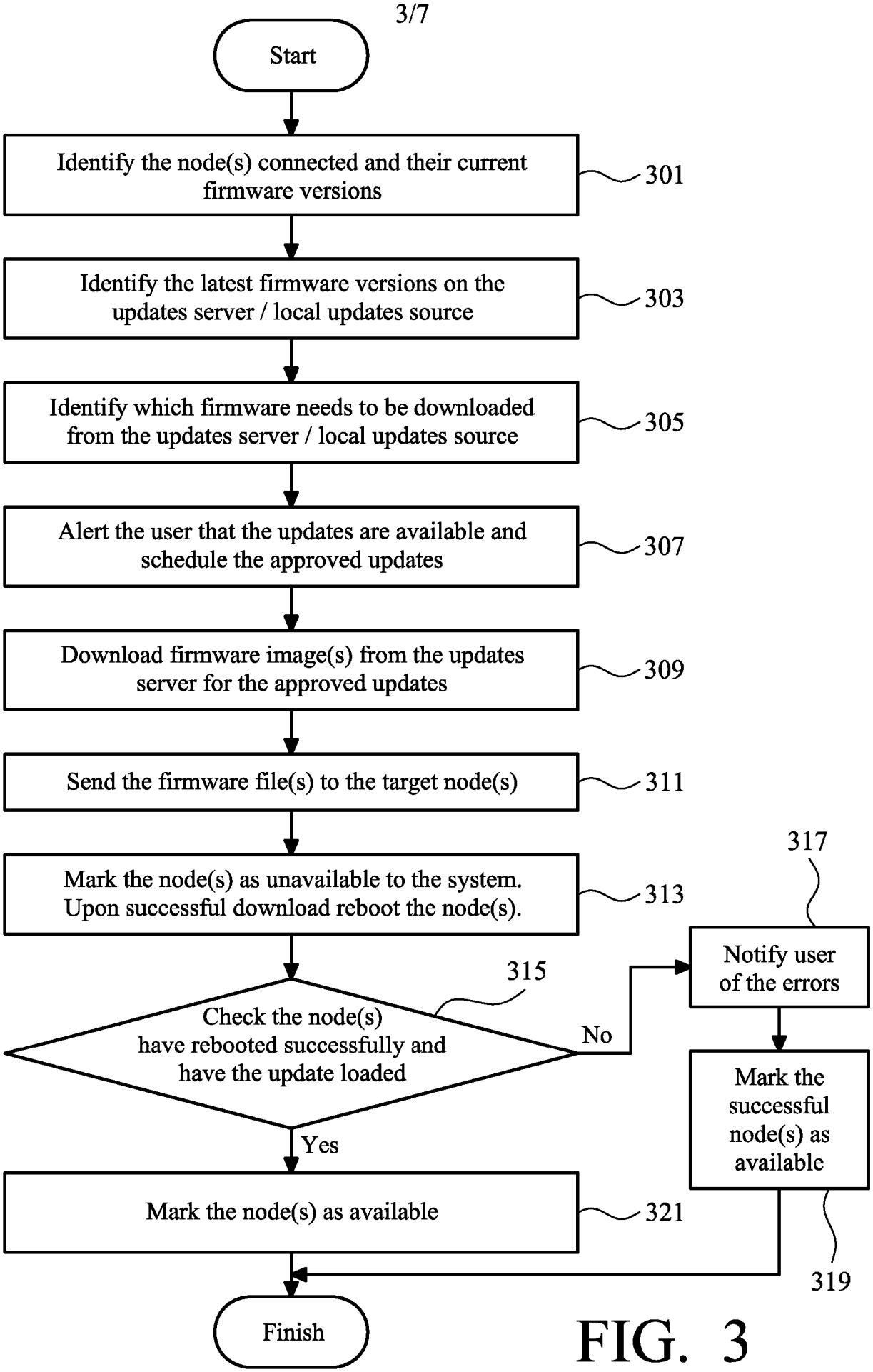
Start
Identify the node(s) connected and their current firmware versions
301
Identify the latest firmware versions on the updates server / local updates source
303
Identify which firmware needs to be downloaded from the updates server / local updates source
305
Alert the user that the updates are available and schedule the approved updates
307
Download firmware image(s) from the updates server for the approved updates
309
Send the firmware file(s) to the target node(s)
311
Mark the node(s) as unavailable to the system. Upon successful download reboot the node(s).
313
315
Check the node(s) have rebooted successfully and have the update loaded
No
Yes
317
Notify user of the errors
Mark the successful node(s) as available
319
Mark the node(s) as available
321
Finish

FIG. 3

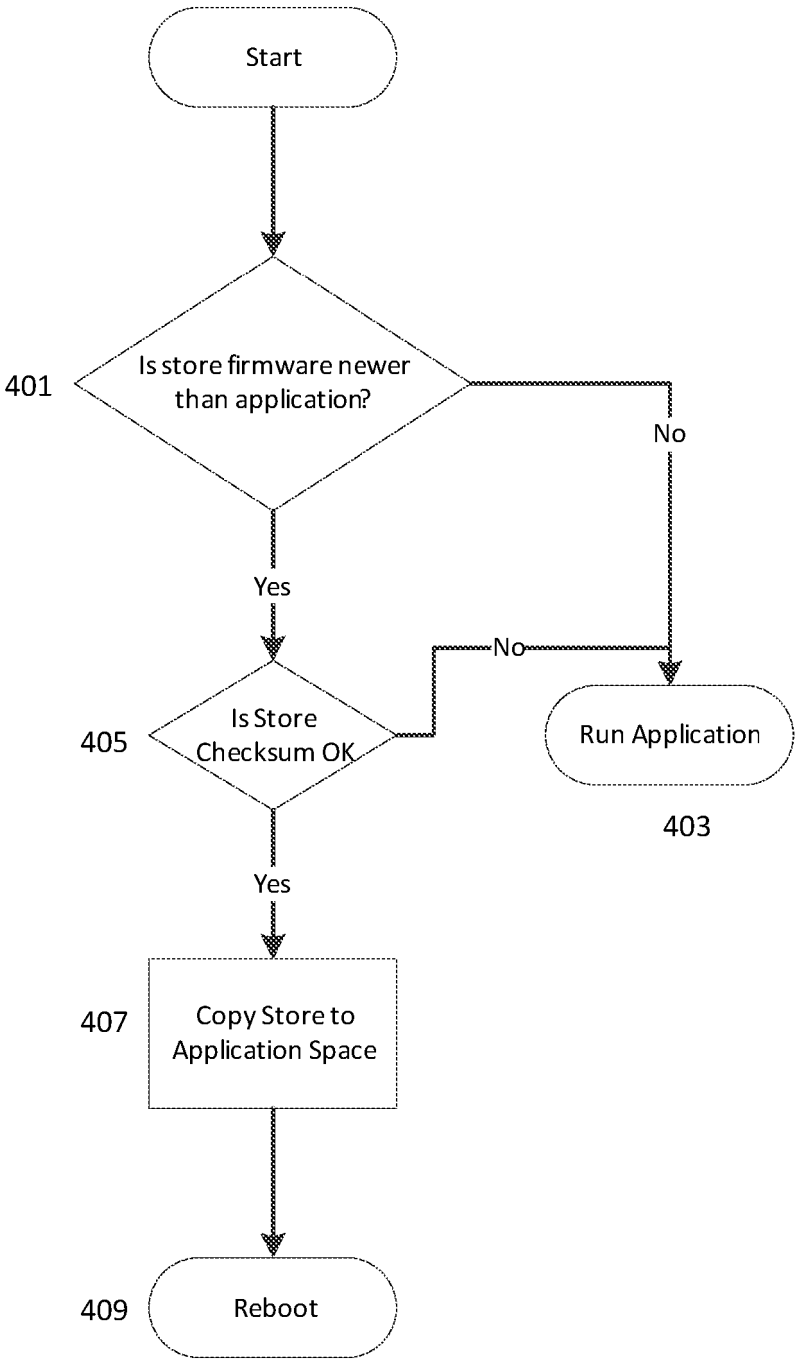
Start
401
Is store firmware newer than application?
No
Yes
No
405
Is Store Checksum OK
Run Application
403
Yes
407
Copy Store to Application Space
409
Reboot

FIG. 4

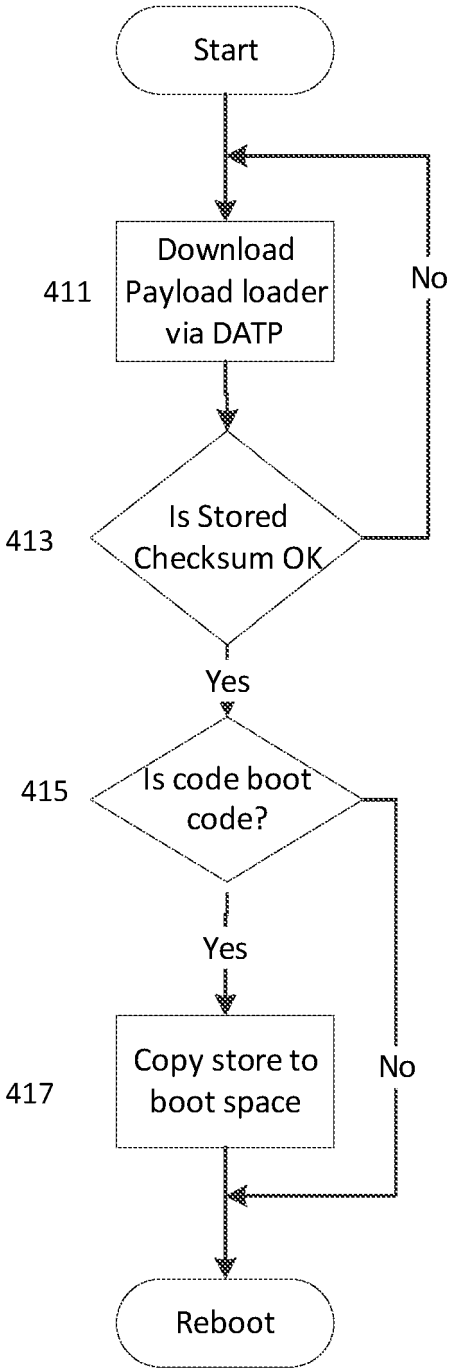
Start
411
Download Payload loader via DATP
No
413
Is Stored Checksum OK
Yes
415
Is code boot code?
Yes
417
Copy store to boot space
No
Reboot

FIG. 5

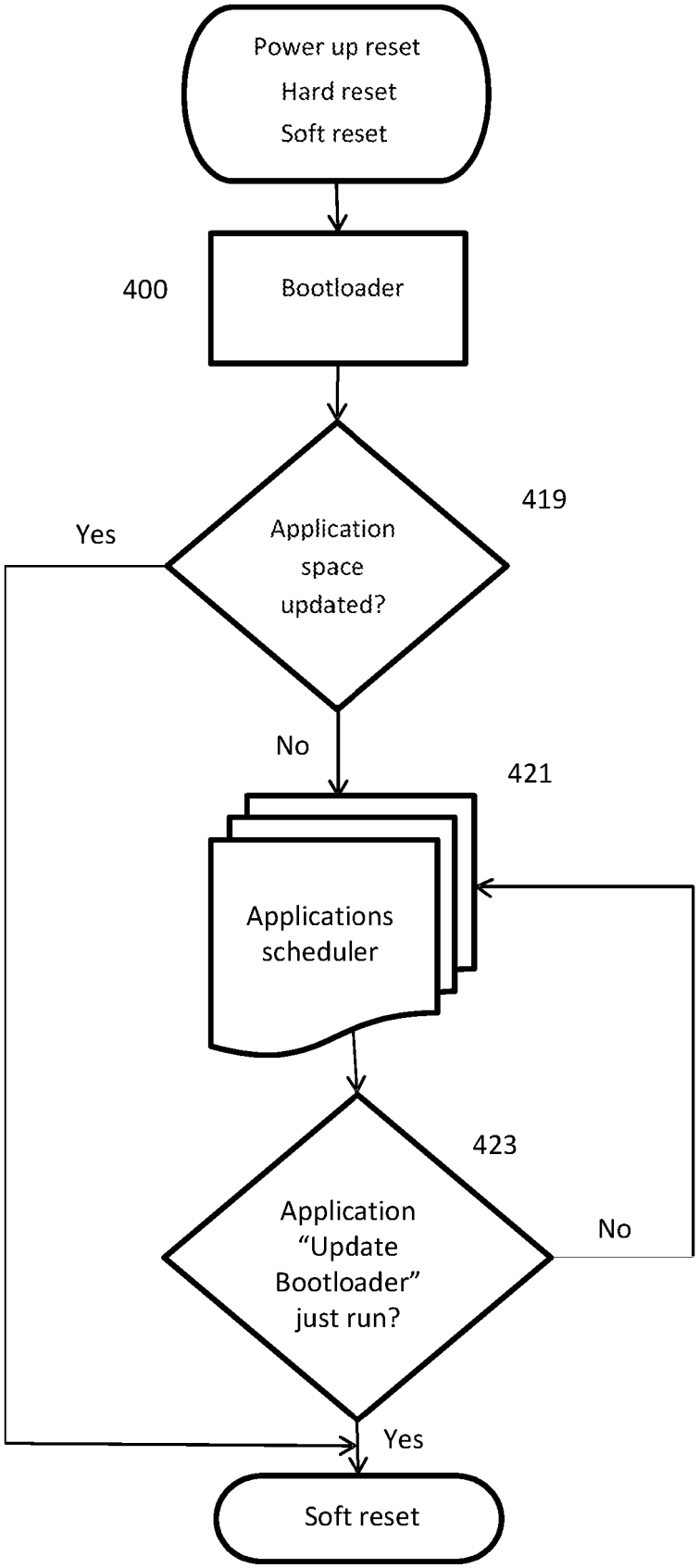
Power up reset
Hard reset
Soft reset
400
Bootloader
419
Application space updated?
Yes
No
421
Applications scheduler
423
Application "Update Bootloader" just run?
No
Yes
Soft reset

FIG. 6

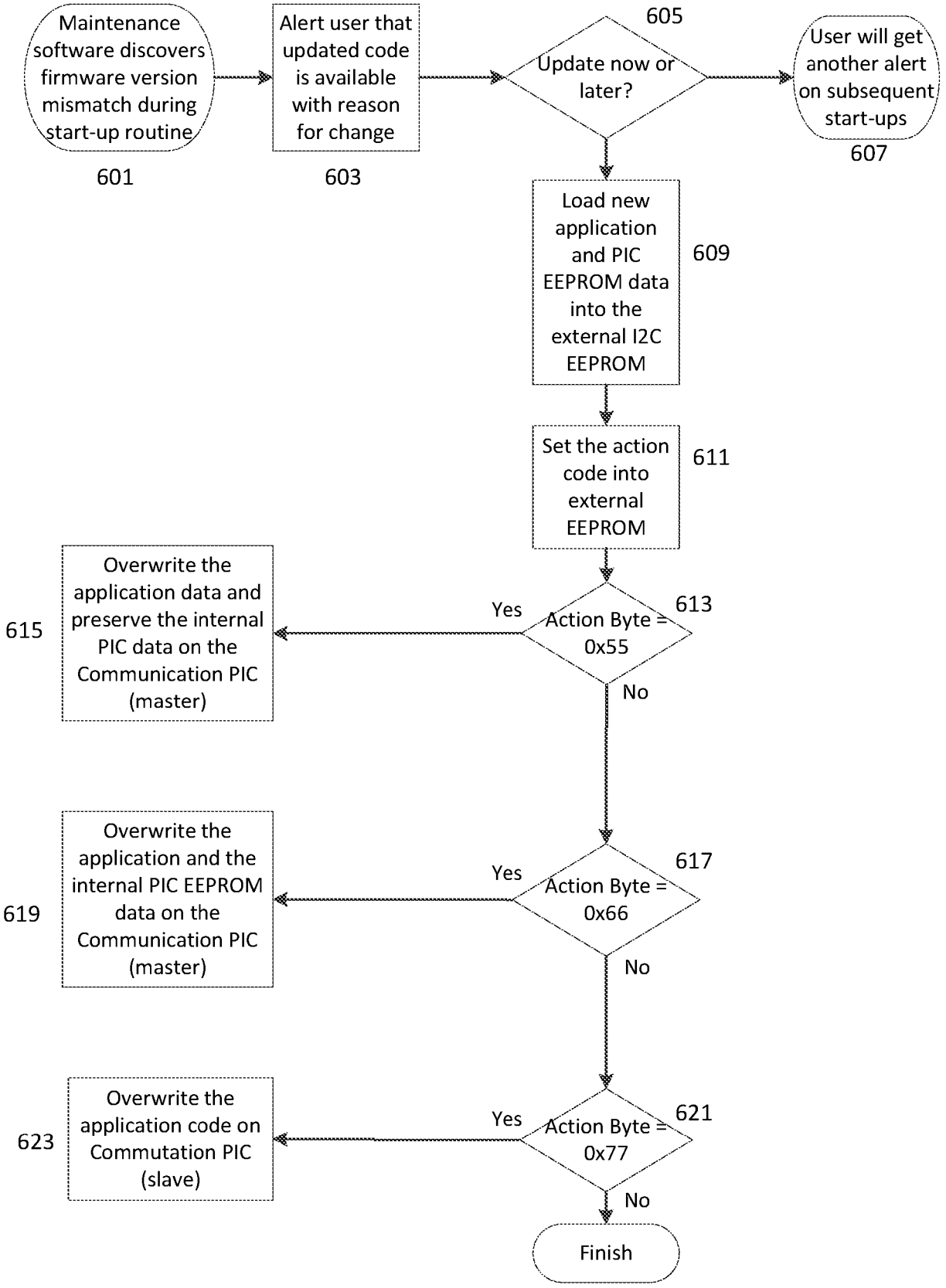
Maintenance software discovers firmware version mismatch during start-up routine
601
Alert user that updated code is available with reason for change
603
605
Update now or later?
User will get another alert on subsequent start-ups
607
Load new application and PIC EEPROM data into the external I2C EEPROM
609
Set the action code into external EEPROM
611
Action Byte = 0x55
613
Yes
Overwrite the application data and preserve the internal PIC data on the Communication PIC (master)
615
No
Action Byte = 0x66
617
Yes
Overwrite the application and the internal PIC EEPROM data on the Communication PIC (master)
619
No
Action Byte = 0x77
621
Yes
Overwrite the application code on Commutation PIC (slave)
623
No
Finish

FIG. 7

The following terms are registered trade marks and should be read as such wherever they occur in this document:

Windows

# METHOD AND APPARATUS FOR UPDATING SOFTWARE ON REMOTELY OPERATED VEHICLE

## Technical Field

The present invention relates to underwater remotely operated vehicles, and particularly to methods and apparatus for updating the software used by components of remotely operated vehicles.

## Background

Remotely operated vehicles (ROVs) are underwater vehicles controlled by an operator on the surface, e.g. on a vessel or other suitable control platform.

Figure 1 shows an example of a known ROV system 101. The main components in a ROV system are a surface power supply unit (109), a control unit (103), a tether (105), and the ROV (107) itself. The control unit (103) is shown with user control interface(s) (111). The tether (105) contains power and data/communication cables that connect to the ROV.

The power supply unit may be connected to an external power supply and provides the system with the various supplies required. The control unit is connected to the power supply unit and contains at least some of the control electronics for the system. The control unit may also provide the outputs from video, sonar and other sensors that may be fitted to the ROV. A video monitor (117) or video suite including a video switcher, video recorders and monitors may be connected to the control unit so that outputs from the ROV systems can be viewed by users on the surface.

ROVs can be "free swimming", in which case the tether connects directly to the ROV. Alternatively a TMS (Tether Management System) (115) may be used, which provides a garage for the ROV and manages the tether to the ROV as it operates. The TMS may be controlled via the control unit (103). The TMS may comprise a remotely controlled tether cable spooling mechanism fitted into the top section of a side entry garage for containing the ROV. The TMS is lowered to the required working depth where the ROV can be flown out of its garage to the extent of its tether cable. A TMS allows various advantages, such as operations to greater depths.

The system comprises several computers, which may range from the control unit through to specific functional modules comprising one or more microcontrollers. The ROV (107) is

laden with multiple different functional modules in the form of components such as propulsion thrusters, cameras, sensors and tools. All of these functional modules communicate with the surface control system via the tether. For example, camera functions such as pan and/or tilt, autopilot functions, thruster trim controls, speed, direction, dive, surface and lighting intensity are controlled via the tether from the surface. Spare capacity for additional control functions may also be included.

The microcontrollers, and other control electronics, for a given functional module may be located within an electronics pod (EPOD) on dedicated hardware, such as one or more control cards or control boards, which are connected to the surface control system via a communications bus. The EPOD is a watertight pod, which may also keep the electronics under vacuum or within a fluid such as oil. The EPOD has standardised electrical connections to service the rest of the on-board components with power and communications. Not all of the functional modules need to include control electronics within the EPOD.

Upgrading the software on an ROV is problematic. As a real-world example, to change firmware in the traditional manner would require the EPOD to be physically removed, the relevant processor cards removed from the EPOD, the ROM(s) reprogrammed, and then the system reassembled. A similar procedure would be required for functional modules within the TMS. Further work is required for components that work in vacuum, or under pressure in oil filled enclosures and the environmental issues that might be caused. This work can only be done at the surface under cover, and can take up to 2-hours under normal conditions.

During use, the ROV can be thousands of meters beneath the sea surface for weeks at a time, and the time to travel between the surface and the working depth is significant and thus expensive. System down time in general is expensive and should be avoided where possible.

An improved way of updating the software on ROV functional modules is required. However, the present inventors have appreciated that such an update mechanism should be resistant to code corruptions caused by unexpected power outages during the update process.

Summary of the Invention

The invention is defined in the independent claims to which reference is now directed. Preferred features are detailed in the dependent claims.

According to a first aspect of the present invention there is provided a method of updating computer executable instructions for one or more functional modules in an underwater remotely operated vehicle (ROV). The method comprises: receiving updated computer executable instructions at a functional module of the ROV from a surface control computer over a communication link; storing the updated computer executable instructions in a first portion of non-volatile memory that is accessible by the processor of the functional module; and subsequently writing the updated computer executable instructions from the first portion of the non-volatile memory to a second portion of non-volatile memory wherein the functional module is configured such that the processor executes computer executable instructions stored in the second portion of non-volatile memory.

By providing a first portion of memory to which the updated computer executable instructions are written before they are actually written to the second memory portion, e.g. a flash ROM, from which the processor fetches its executable code, it is possible to ensure that the updated code has been received in its entirety before there is a possibility of the processor executing it. This prevents the processor attempting to execute incomplete or corrupted code. As such, if a power outage occurs during the process of downloading the updated executable code, or instructions on how to update the existing code, the functional module will always have the pre-existing executable code already loaded in the firmware memory.

Optionally, the method further comprises determining whether updated computer executable instructions are stored in the first portion of non-volatile memory, wherein: if updated computer executable instructions are stored, then the update procedure proceeds by writing the updated executable code from the first memory portion to the second memory portion; and if updated computer executable instructions are not stored, the method further comprises controlling the processor to execute pre-existing executable code in the second memory portion. Optionally, determining whether updated computer executable instructions are stored in the first portion of non-volatile memory comprises: executing, on the processor, first predetermined instructions according to an "action" code stored in a portion of non-volatile memory accessible by the processor, the action code being indicative of whether updated computer executable instructions are stored, wherein: if updated computer executable instructions are stored then the action code causes the processor to update the computer executable instructions by writing the executable code from the first portion of the non-volatile memory to the second portion of non-volatile

memory; and if updated computer executable instructions are not stored then the action code causes the processor to execute pre-existing executable code in the second memory portion. Optionally the method may further comprise setting the action code to cause the processor to execute pre-existing executable code in the second portion of memory only when updating the computer executable instructions has completed.

The check for updated executable code actually involves the processor looking to the first portion of non-volatile memory. For example, this may happen automatically upon boot of the processor. The instructions contained within, or identified by, the first portion of memory cause the processor to either over-write the executable code in the second portion of memory (e.g. the firmware flash ROM) if there is an update to apply, or to proceed as normal and to execute existing executable code if there is no new update to apply.

Furthermore, the update process can be controlled by setting the action code as appropriate, such that booting to, or executing, updated executable code will only occur once the updated executable code has been completely written to the second memory portion (and optionally verified). Otherwise, the processor will attempt to retrieve updated code from the first memory portion. By ensuring that the action code is only set to cause execution of the executable code in the second portion of memory once writing from the first memory portion to the second memory portion has completed it is ensured that the processor will not attempt to execute corrupted code. If a power outage occurs during the update process, particularly when the executable code on the second memory portion is being over-written, the update procedure will simply resume when power is restored, with the processor looking again to the first memory portion to see if new code is available. In this way, the processor will always execute either completely updated executable code or an earlier version of the executable code, and will not attempt to execute corrupted code.

Optionally the method may further comprise performing a first check to verify the updated computer executable instructions stored in the first portion of non-volatile memory. If the first check fails then writing of the updated computer executable instructions from the first portion of the non-volatile memory to the second portion of non-volatile memory may be prevented. After the first check fails, the method may further comprise repeating the steps of receiving the updated computer executable instructions at the functional module, storing the updated computer executable instructions in the first portion of non-volatile memory, and checking the updated computer executable instructions according to the first check; whereby writing of the updated executable code from the first portion of the non-volatile memory to the second portion of non-volatile memory is permitted only when the first check is passed. The check is performed using a cyclic redundancy check (CRC) method for example.

By performing a check to verify the updated executable code stored in the first portion of non-volatile memory it is possible to ensure that the correct, uncorrupted, updated executable code has been received before it is loaded into the firmware memory (i.e. the second memory portion). By preventing the writing of code from the first memory portion to the second memory portion until the verification of the downloaded code is satisfied, it is possible to ensure that the received code has not been corrupted, e.g. due to a power outage, during the download process. If the check fails then repeating the download of code, storing it to the first portion of memory and checking it, until the check passes, ensures that code will only be downloaded into the firmware memory when download has been successful, preventing the processor trying to execute corrupt code.

Optionally, the ROV comprises multiple functional modules, and the method further comprises updating one functional module at a time by sending the entire set of updated computer executable instructions from the surface control computer to each module in turn. Alternatively, the method may further comprise updating the multiple functional modules simultaneously by sending portions of the updated computer executable instructions from the surface control computer, each portion addressed to a respective functional module, in a predetermined sequence. Alternatively the method may further comprise updating multiple functional modules simultaneously by broadcasting the updated computer executable instructions to multiple functional modules of the same type.

Optionally the first memory portion contains multiple versions of computer executable instruction sets, and the method further comprises selecting which of the sets of multiple versions of computer executable instructions is to be written to the second portion of memory. The sets of multiple versions of computer executable instructions may be selectable by: executing, on the processor, first predetermined instructions according to an "action" code stored in the first portion of non-volatile memory, the action code being indicative of the version of the executable code to be selected, wherein: the action code causes the processor to write the selected executable code from the first portion of the non-volatile memory to the second portion of non-volatile memory for subsequent execution by the processor. This allows different versions of firmware to be loaded, providing different functional settings depending on the operational requirements of the ROV.

Optionally, each of the one or more functional modules is a node that communicates with the control computer according to a communication protocol and is a single addressable end point.

Optionally, at least one of the one or more functional modules is a master functional module coupled to one or more slave modules, and the method further comprises: writing updated computer executable instructions from the first portion of the non-volatile memory to a third portion of non-volatile memory, wherein the slave module is configured such that a processor of the slave module executes computer executable instructions stored in the third portion of non-volatile memory. Optionally the method further comprises setting an action code to cause the processor of the master module to execute pre-existing computer executable instructions in the second portion of memory only when writing the updated computer executable instructions from the first portion of the non-volatile memory to the third portion of non-volatile memory has completed.

Optionally the first portion of non-volatile memory is on a first non-volatile memory unit and the second portion of non-volatile memory is on a second distinct non-volatile memory unit. The first memory may be one of an EEPROM or an SD card. The second memory portion may be a flash ROM.

The computer executable instructions may be firmware or boot code.

Optionally the communication link includes a wired connection that passes through a tether connecting the surface control computer to the ROV or to a Tether Management System, the tether containing power and data cables.

Optionally the communication link includes a wireless connection.

A corresponding functional module for an underwater remotely operated vehicle (ROV) may also be provided, along with an underwater remotely operated vehicle (ROV) comprising one or more such functional modules, and a system comprising a control computer and a corresponding ROV.

A computer program may also be provided that when executed on a functional module causes it to carry out the methods described herein.

A surface control computer may also be provided that provides updates to one or more functional modules in an underwater ROV.

Where functional modules are provided in the TMS, any embodiments described herein may be applied equally to those functional modules.

Brief Description of the Drawings

Examples of the invention will now be described in more detail with reference to the accompanying drawing in which:

Figure 1 is an overview example of a known ROV system;

Figure 2 is an overview example of an ROV system in accordance with embodiments of the invention;

Figure 3 is an example of the primary computer control system flow for the update procedure, showing an example of how to send the firmware update to the appropriate functional modules;

Figure 4 is an example update procedure for application code/firmware;

Figure 5 is an example update procedure for boot code;

Figure 6 is an example update procedure for updating either firmware or boot code; and

Figure 7 is an example update procedure using action codes or bytes.

Detailed Description of Preferred Embodiments

Figure 2 shows an example of an ROV system 201 used in accordance with embodiments of the present invention. Components in common with Figure 1 are given the same reference numerals.

To improve upgrades to ROV systems a DATP (Data and Application Transfer Protocol) is provided. The DATP enables software and configurations to be transferred via telemetry to ROV components over the tether communication lines without the need to remove the components from the ROV, or even bring the ROV to the surface.

In embodiments of the invention a software or firmware update may be transferred using a sub-protocol embedded within the system's existing telemetry. The transfer is sent over telemetry and stored on a first portion of non-volatile memory, where it may be verified and checked, and if it is a different version from that installed then the firmware will be read from the first portion of memory and installed to a second portion of memory from which the processor executes executable code. The first portion of non-volatile memory may be an external media such as an SD card, I2C EEPROM, and so on. The second portion of

memory may be the firmware memory, such as a flash memory, from which the processor executes code when the system reboots.

A control computer (113) is provided, which acts as the control hub for the whole system. As before, the system comprises several computers, which may range from PCs (personal computers) such as the primary control computer, through to specific functional modules comprising one or more microcontrollers.

Any of the electronic/electromechanical components on the ROV (207), ranging from lamps, hydraulic pumps, cameras, control arms and communication modules, could have one or more microcontrollers inside. Any module that contains intelligence for carrying out one or more ROV functions may be considered a functional module. Functional modules, in most practical embodiments, will be implemented as a programmed board of some description having a microcontroller.

Any ROV functional module which hosts a communication protocol allowing communication between the functional module and the control system via the tether may be referred to as a "node". A node may, in particular, form a single end point that the control system can address, or to which the control system can communicate. The nodes are all connected via a common communication network and protocol. For example, the nodes may be connected over a mix of physical interfaces using the UDP (User Datagram Protocol) transport layer.

Each node may have a unique ID number which can be used to address it. The control computer (113) may then maintain a list of node IDs in the system. Third party or peripheral components may still be addressed as a node by virtue of an interface PCB being fitted.

Each of the nodes includes one or more microcontrollers having one or more processors for executing executable code to cause the node to carry out its function within the ROV. The control electronics for a given node may be housed within an EPOD of the ROV, or they may be housed elsewhere on the ROV. A given EPOD for an ROV may, for example, contain around 30 nodes. The control electronics may be provided on one or more control cards or control boards.

Each of the nodes has non-volatile memory provided specifically for the purpose to download updated executable code into. The updated executable code may be a new firmware image for example. This makes the upgrade robust against the updated

executable code being corrupted and leaving the microcontroller with an incomplete or corrupted firmware from which it could not recover.

Figure 3 shows an example of the primary computer control system flow for the DATP. The method shown in Figure 3 allows the control computer (113) to send the firmware update to the appropriate functional modules.

At step 301 the control computer identifies the node(s), or functional modules, connected within the ROV and their current firmware versions. At step 303 the control computer then identifies the latest firmware versions available.

The firmware updates may be provided to the control computer in a number of different ways. For example, updates may be provided from an updates server (119) over the internet and/or one or more local area networks. Alternatively, or in addition, the updates could be provided locally to the control computer via one or more local update sources (121).

At step 305 the control computer identifies which firmware an update is available for, and therefore needs to be downloaded from the updates server/local updates sources. The user may then be alerted that updates are available, and approved updates scheduled. Alternatively, the update process may be automatic.

At step 309 the required firmware is obtained from the updates server, or the local updates sources, as required.

At step 311 the relevant firmware files are sent to the target functional modules over the communication network, via the tether communication line(s) of the tether (105).

The target functional modules then implement their local update procedure, which will be detailed further below. During this procedure, after the update file(s) have been sent to the target functional modules the relevant target functional modules are marked as unavailable to the ROV system (101) by the control computer. After a successful update procedure each of the target functional modules is rebooted. At step 315 a check is made by the control computer that each of the target functional modules has successfully rebooted, and that the update has successfully loaded into the firmware memory. If this check proves to be negative then, at step 317, the user is notified of the errors and any functional modules where the update was successful are marked as available. If all the functional modules are successfully updated then they are all marked as available at step 321.

The procedure carried out at each functional module will now be described. In general, three steps are carried out during the update.

In a first step, the firmware update file is stored in an external memory such as EEPROM on the control card of the functional module. A check, such as a CRC or checksum, may also be performed as part of the first step. Only when the first step is complete does the second step commence.

In a second step the microcontroller, e.g. a Programmable Intelligent Computer (PIC), of the functional module is reset. This may be a software reset, since the purpose of the reset is simply to cause a re-boot.

In a third step, during re-boot, the bootloader of the microcontroller causes the microcontroller to perform a check for a new update in the EEPROM. This check may be based upon firmware version numbers, although in some embodiments the check is performed by either executing an instruction to update from EEPROM, or executing an instruction to boot from existing firmware, with the instruction being specified by a reference or code within the EEPROM. If new firmware has been received then it is copied to the internal memory, e.g. a flash memory, of the functional module that stores the firmware executed during operation.

Two example methods will be used to describe example embodiments.

According to a first method, Figure 4 shows an example of the boot process, employed by the bootloader. The bootloader is the first code to run after reset or power-up. It is designed to be very small and limited in function, in the hope that it will never change and is bug free.

The bootloader can be used to update application code/firmware by copying the code stored in the external memory to the internal memory used for the application code. The application code cannot do this as it would be overwriting itself. The application code can then be run if applicable.

At step 401 a check is performed to determine whether the stored firmware stored on the EEPROM is a later version of the firmware currently stored on the firmware flash memory of the functional module. If the stored firmware is not newer than that already loaded into the flash memory then the previously stored existing firmware is executed at step 403.

If a new version of the firmware is stored in the EEPROM then a verification step may be performed, such as a check sum or CRC at step 405. If the verification fails then the pre-stored firmware already loaded into the functional modules flash is executed. If the check passes then the updated firmware is copied from the EEPROM to the flash memory of the functional module that stores the firmware executed during operation. At step 409 the functional module microprocessor then reboots. After reboot the process may return to step 401, whereby after an update to the latest firmware version the procedure will move to step 403 and the newly updated firmware will be executed.

The steps of Figure 4 are performed in one or more functional modules/nodes of the system as appropriate.

Figure 5 shows an example of a process that may be used to update the boot loader. This process may be used in conjunction with the process described for Figure 4, and may be performed by an application executing on the functional module/node.

The payload, containing the bootloader update code, is downloaded at step 411 to external memory/EEPROM. A check 413 may be performed, e.g. using a checksum, in order to confirm that the code was downloaded correctly.

At step 415 a check is performed to determine whether the received payload includes boot code, as opposed to other types of code. If the code received is boot code then the code is copied, at step 417, to the boot space memory portion from which the node/module processor executes boot code.

Figure 6 shows an example of the overall process that might be implemented upon power-up or reset of a given functional module or node, which provides a way of updating both application firmware and boot code.

Step 400 represents the bootloader process, whereby application software updates may be checked for and applied as necessary. An example of the process 400 is given in Figure 4, and described above.

A check may be performed at step 419 to determine whether the application storage space that stores the firmware/application code executed during operation has been updated with new application firmware/software. Where this is the case, a reset or restart may be applied and the relevant checks performed again.

The Application Scheduler 421 is the process by which all the application code is run. The Application Scheduler determines when individual applications run. Examples of Applications could include: Flash LED, receive serial message, send serial message, interpret message and an "Update Bootloader" application, an example of which is shown in, and described in relation to, Figure 5.

When the Application Scheduler determines that the "Update Bootloader" application described in relation to Figure 5 is to be run, for example due to an appropriate command (e.g. a DATP command) being detected, the application runs and in the first 2 steps of Figure 5, the new boot image is downloaded. An advantage of doing this at application level is that the hardware interface to the communication ports and the external memory (e.g. the EEPROM) may use the application library functions and leave the bootloader very simple.

On successful completion, as described for Figure 5, the process then checks 415 if the received code is boot code for the bootloader. This should be a rare occurrence. If it is then it copies 417 the bootloader from external memory to internal memory. It can do this as this process is running in application space so the bootloader is free to be over-written.

If the downloaded code is not boot code then the module processor is rebooted / reset. This takes the method back to the bootloader process, which detects if there is a valid newer version of the application space/firmware in the external memory (e.g. EEPROM) and copies this to the internal application space of the functional module that stores the firmware executed during operation. The process reboots or restarts one more time, after which the bootloader will not find a new version of application code (if no further updates have issued), and so then immediately runs the main application scheduler to run other applications.

As an example, the method described in relation to Figure 4, Figure 5 and/or Figure 6 may be applied in relation to a particular example system architecture, referred to as "digital", which is effectively a serial full duplex, direct command strings, protocol between the surface control computer and a computer/microcomputer within the ROV. The ROV computer then communicates directly via serial lines to the functional module endpoints in the ROV. This is an example of a single end to end architecture, in which all ROV processing may be performed on a single subsea processor, in which all control algorithms (e.g. auto-heading, auto-depth, auto-altitude, gyro response etc) are executed subsea.

Figure 7 shows the DATP flow according to a second exemplary method. At steps 601 to 607 a series of steps may optionally be performed depending upon whether updates are to occur manually, under instruction from a user. At step 601 software executing on the control computer identifies firmware mismatches between the latest available firmware and the firmware loaded on one or more functional modules. At step 603 an alert may optionally be provided to the user indicating that an update to the firmware is available. At step 605 a decision is taken, by the user or otherwise, to update now or at a later date. If a later update is requested then the user will get a further alert 607 on subsequent start ups of the system indicating that updates are still available.

At step 609 the downloaded firmware updates are loaded into respective first memories (e.g. EEPROMs) located on each of the respective functional modules that are being updated. In the case of step 609 the system performs the step of storing the code into the first portion of memory (e.g. the external EEPROM). Upon successful completion there is a command to set one or more action codes.

The one or more action codes may be used to determine the actions carried out by the functional module processor. At step 611 the action code or codes is/are set into the first portion of memory (e.g. the external EEPROM). The action codes may be separate commands provided in a manner other than the firmware update download.

The action codes are tokens to indicate the action that is required to happen next. The action codes are automatically read by the functional module processor, e.g. at steps 613, 617 or 621 in Figure 7, for example upon boot up. A valid action could include, for example, to overwrite the firmware/application code. The action codes may be stored in the first portion of memory, e.g. the EEPROM, although they could be stored in any accessible non-volatile memory that is not going to be overwritten. The action codes may be numeric values. The action codes may identify code stored in the second portion of memory (e.g. the flash memory) that causes the processor to execute the action associated with the action codes.

Three examples of relevant action codes are given in Figure 7. One or more of these action codes may be used in combination to achieve the desired updating of application code and/or boot code in relation to a master or slave processor. Examples of action codes are provided below.

If the action code is the sort indicated by the action byte identified at step 613 then this will cause, at step 615, the functional module processor to overwrite the firmware data from the

EEPROM to the flash memory. However, the communication protocol data used by the functional module may be handled separately, outside the bootloader program, and so therefore may be specifically preserved rather than being overwritten when a firmware update occurs.

If the action code is that identified by the action byte of the sort shown in step 617 then, at step 619, both the bootloader used by a master functional module is overwritten on the flash memory of the master functional module, as well as, optionally, the application code.

If the action code is that identified by the action byte at step 621 then, at step 623, the firmware or application on a slave functional module is overwritten.

As mentioned above, the action codes, which may be used in any embodiments of the invention, may be provided in the code stored in the first portion of non-volatile memory, e.g. the EEPROM. The processor, when performing a boot, will automatically look to the action code on the EEPROM. The action code may provide executable code, or a reference to/address for executable code within the boot loader to carry out a particular task. The task may be for the processor to boot from, or execute the firmware already contained in, the functional module's flash memory. Alternatively the action code may be set to cause the processor to overwrite the firmware and/or boot code in the flash memory.

The action code can be set accordingly so that the processor carries out the desired function upon boot. It is only when the firmware has been overwritten into the flash memory that the action code is changed such that upon subsequent boots the processor does not look to the EEPROM for updated firmware, but instead looks to its own flash memory. By updating the action code last, after the firmware has been written to the flash memory, the module processor is prevented from attempting to run corrupt code that has not fully been received (for example when power is lost during transfer).

The second exemplary method may be applied within an architecture that uses a control system application running on an operating system, such as Windows (TM) on the control computer (113).

The architecture of the second exemplary method may be a "multidrop" nodal architecture, in which each node handles its own functions. All control algorithms (e.g. auto-heading, auto-depth, auto-altitude, gyro response etc) may be executed from the surface control computer processor. An example of such an architecture may be referred to as iCON (Intelligent Control of Nodes).

The application software used in the second exemplary method may be provided using a framework and a kit of software programs that can be assembled to form a user interface and high-level control system of an ROV system. The various programs are selected as needed to develop the desired functionality and cover a wide variety of functions such as UI graphical representations of components, a program that allocates thrust demands to the thrusters, a program to read data from a sensor, and so on. The programs are configured to form subsystems, e.g. by means of configuration files. The framework is responsible for loading and managing the required components. Communication within the various sub-systems may be performed using the OPS (Open Publish-Subscribe) protocol for example. As an example, the functional modules/nodes of the ROV may run an embedded application layer for most of the functions (thrusters, lights, power channels etc). Translation from the protocol used for communication between the various sub-systems and the embedded application layer for the nodes may be provided as appropriate. For example, a Castle embedded application layer may be used for the functional nodes, which may then be translated to by another layer (e.g. a Rasputin layer) from OPS on the control computer (113). It should be noted, however, that translation is not always required, for example some traffic can go directly from the control computer (113) through OPS, bypassing Rasputin, directly to the nodes.

In any embodiment any of the functional modules/nodes may be arranged in a master/slave arrangement, whereby a given functional module has one or more slave functional modules that look to a master functional module.

An example of a master/slave arrangement would be thruster functional modules, in which the master functional module communicates with various sensors on the ROV thrusters and the slave is configured simply to drive the thruster motors. In this context the master functional module may send signals to the slave functional module indicating how fast the slave should spin the motor. The slave controls the motor to spin at the desired rate, and communicates back to the master to tell it how fast it is spinning. Other data may also be shared such as fault conditions.

In order to update the slave firmware the master functional unit's processor may overwrite data from the first portion of non-volatile memory, e.g. the EEPROM, on the master functional node to the flash memory on the slave functional node, without the slave functional node requiring its own EEPROM (although it is also a possibility that the slave would have its own EEPROM, or equivalent, and update in the other manners described herein).

Optionally, when updating a slave module using a method involving action codes, such as the method described in relation to Figure 7, action codes may not be changed in the master EEPROM from a first action code, that causes the processor to overwrite the firmware on the flash memory, to a second action code, that causes the processor to execute firmware stored in the flash memory, until overwriting of the updated firmware on the flash memory of the slave functional module is complete.

In the case of a functional module or node having multiple microcontrollers, one microcontroller can act as a master to several slave microcontroller devices to update the firmware.

Generally, where the ROV comprises multiple functional nodes, only one node at a time, or one set of nodes of a particular type (e.g. thrusters) at a time, may be updated with new application/boot code. The method may then further comprise updating one functional module, or a set of functional modules of the same type, at a time by sending the entire updated executable code from the surface control computer to each module, or set of modules, in turn.

Alternatively, a plurality of nodes of different types may be updated simultaneously by sending different portions of code over the same transmission, with different portions being sent to different nodes. In particular, portions of the updated executable code may be sent from the surface control computer, each portion addressed to a respective functional module, in a predetermined sequence.

As an alternative method, the update data may be broadcast to multiple nodes/modules of the same type (e.g. thrusters), so that they receive the same executable code and are updated together. Because the update data is a broadcast, the modules cannot acknowledge the messages and some may fail silently. Optionally, the surface control computer may poll each functional module at the end of the broadcast to determine whether the update was successfully received and/or applied. Any failed updates can then be fixed individually if necessary. Embodiments employing such an update strategy are of most use when there are many devices/modules of the same type, as they all can be updated in a time frame slightly greater than that needed to update a single module.

Throughout the detailed description the example of performing a firmware update has been used. It will be appreciated that the methods described herein can be applied to any type

of executable code stored on a memory for execution by a functional module, also including the bootloader.

Embodiments of the invention have generally been described using different memories for the first and second memory portions, the first portion being provided to receive updated code or instructions, and the second being the internal memory used to store instructions executed by the processor of the functional module. If is, however, possible to instead use a single memory for both purposes if a large enough memory can be provided. Where a single memory is used, a first dedicated memory portion is provided for storing updated code or instructions, and a second dedicated memory portion is provided for storing processor instructions.

Embodiments of the invention have been generally described that use a communication bus, or more generally a communication link, that passes through the tether connecting the surface control computer to the ROV, or to the tether management system or ROV garage. However, it is also possible for the ROV to operate wirelessly, whereby all or part of the communication link is a wireless connection. For example, the ROV may communicate with an underwater communication system that provides a communication link to the surface (e.g. via tether). The communication between the ROV and the communication system may be via a wireless communication link, using short range light modems and/or acoustic modems to a modem head, or receiver, for example. The communication system may also function as an underwater, or seabed, docking station. The docking station may allow the ROV to dock to recharge, and may also provide a direct high speed communication link to the surface computer. The docking station may be linked by a high data rate physical medium to a local surface structure or vessel, e.g. an oil rig, where the usual surface control is available.

Embodiments have generally been described in which computer executable instructions are received at, or written to, a first memory portion for copying across to a second memory portion in order to update pre-existing computer executable instructions on an ROV functional module. However, according to another aspect of the invention, for some ROV functional modules the process of updating the functional module may instead comprise following a set of instructions to reprogram a programmable device to do so according to an update file. Therefore, rather than necessarily receiving updated computer executable instructions from a surface control computer, instead an update file to update the functional module may be received at the functional module of the ROV from the surface control computer over the communication link and stored in the first memory portion. The update file, when executed by a module processor, causes the update of the functional module.

In general, according to this aspect, a method of updating one or more functional modules in an underwater remotely operated vehicle (ROV) is provided, the method comprising: receiving an update file at a functional module of the ROV from a surface control computer over a communication link; storing the update file in a first portion of non-volatile memory that is accessible by the processor of the functional module; and subsequently writing the update file from the first portion of the non-volatile memory to a second portion of non-volatile memory, wherein the functional module is configured such that a reconfigurable device is updated as specified by the update file. This aspect may be combined with the various optional features described above for the embodiments in which executable code, or firmware, is replaced with new code to perform the update. The reconfigurable device may be updated using the update file in the second memory portion, for example by the functional module processor, or another processor of the module, accessing the update file in the second portion and performing the updating according to the update file information.

As an example, the functional modules to be updated may include programmable logic devices, or reconfigurable devices, such as FPGAs (Field Programmable Gate Arrays), CPLDs (Complex Programmable Logic Devices), PLCs (Programmable Logic Controllers) and so on. For these types of programmable devices, machine instructions might not be loaded for execution, but rather fuses, or equivalent, are programmed to configure the logic of the device as required by the update file.

Such embodiments can be implemented in the same ways as providing updated executable code as described above, but the data accompanying the executable code (which might include look-up tables, configuration settings or fonts for example) includes relevant update data such as fuse data for FPGAs etc.

The update file may be received at the functional module. The update file may, in the case of a FPGA for example, be a fuse file indicating which fuses to activate to cause the desired changes to the FPGA functionality. For example, for FPGAs the fuses are usually arranged in a linear list that define some or all fuses to be set. These may be one time fuses, or the fuses may be RAM based.

The update file may be downloaded to the first memory portion (e.g. the EEPROM), and then be transferred to a second memory portion (e.g. the processor flash memory). Alternatively, the update file may be transferred to another memory portion dedicated to the programmable device, such as a flash dedicated to the FPGA.

The programmable device, such as an FPGA, might therefore have its own memory portion, such as an external or internal EEPROM. The boot loader might program it directly based on the update file. The update file (e.g. fuse file) may be provided to the FPGA EEPROM directly, or may be provided to the module processor's first memory portion (e.g. EEPROM) as described above, and then provided to the dedicated memory portion of the programmable device for subsequent update.

CLAIMS

1. A method of updating computer executable instructions for one or more functional modules in an underwater remotely operated vehicle (ROV), the method comprising:

- receiving updated computer executable instructions at a functional module of the ROV from a surface control computer over a communication link;

- storing the updated computer executable instructions in a first portion of non-volatile memory that is accessible by the processor of the functional module; and

- subsequently writing the updated computer executable instructions from the first portion of the non-volatile memory to a second portion of non-volatile memory wherein the functional module is configured such that the processor executes computer executable instructions stored in the second portion of non-volatile memory.

2. The method of claim 1 further comprising determining whether updated computer executable instructions are stored in the first portion of non-volatile memory, wherein:

- if updated computer executable instructions are stored, then the update procedure proceeds by writing the updated executable code from the first memory portion to the second memory portion; and

- if updated computer executable instructions are not stored, the method further comprises controlling the processor to execute pre-existing executable code in the second memory portion.

3. The method of claim 2 wherein determining whether updated computer executable instructions are stored in the first portion of non-volatile memory comprises:

- executing, on the processor, first predetermined instructions according to an "action" code stored in a portion of non-volatile memory accessible by the processor, the action code being indicative of whether updated computer executable instructions are stored, wherein:

- if updated computer executable instructions are stored then the action code causes the processor to update the computer executable instructions by writing the executable code from the first portion of the non-volatile memory to the second portion of non-volatile memory; and

- if updated computer executable instructions are not stored then the action code causes the processor to execute pre-existing executable code in the second memory portion.

4. The method of claim 3 further comprising setting the action code to cause the processor to execute pre-existing executable code in the second portion of memory only when updating the computer executable instructions has completed.

5. The method of any preceding claim further comprising performing a first check to verify the updated computer executable instructions stored in the first portion of non-volatile memory.

6. The method of claim 5 wherein if the first check fails then writing of the updated computer executable instructions from the first portion of the non-volatile memory to the second portion of non-volatile memory is prevented.

7. The method of claim 6 wherein the method further comprises:

- after the first check fails, repeating the steps of receiving the updated computer executable instructions at the functional module, storing the updated computer executable instructions in the first portion of non-volatile memory, and checking the updated computer executable instructions according to the first check;

- whereby writing of the updated executable code from the first portion of the non-volatile memory to the second portion of non-volatile memory is permitted only when the first check is passed.

8. The method of any of claims 5 to 7 wherein the check is performed using a cyclic redundancy check (CRC) method.

9. The method of any preceding claim wherein the ROV comprises multiple functional modules, the method further comprising updating one functional module at a time by sending the entire set of updated computer executable instructions from the surface control computer to each module in turn.

10. The method of any of claims 1 to 8 wherein the ROV comprises multiple functional modules, the method further comprising updating the multiple functional modules simultaneously by sending portions of the updated computer executable instructions from the surface control computer, each portion addressed to a respective functional module, in a predetermined sequence.

11. The method of any of claims 1 to 8 wherein the ROV comprises multiple functional modules, the method further comprising updating multiple functional modules simultaneously by broadcasting the updated computer executable instructions to multiple functional modules of the same type.

12. The method of any preceding claim wherein the first memory portion contains multiple versions of computer executable instruction sets, the method further comprising selecting which of the sets of multiple versions of computer executable instructions is to be written to the second portion of memory.

13. The method of claim 12 wherein the sets of multiple versions of computer executable instructions are selectable by:

- executing, on the processor, first predetermined instructions according to an "action" code stored in the first portion of non-volatile memory, the action code being indicative of the version of the executable code to be selected, wherein:

- the action code causes the processor to write the selected executable code from the first portion of the non-volatile memory to the second portion of non-volatile memory for subsequent execution by the processor.

14. The method of any preceding claim wherein each of the one or more functional modules is a node that communicates with the control computer according to a communication protocol and is a single addressable end point.

15. The method of any preceding claim wherein at least one of the one or more functional modules is a master functional module coupled to one or more slave modules, the method further comprising:

- writing updated computer executable instructions from the first portion of the non-volatile memory to a third portion of non-volatile memory, wherein the slave module is configured such that a processor of the slave module executes computer executable instructions stored in the third portion of non-volatile memory.

16. The method of claim 15 further comprising setting an action code to cause the processor of the master module to execute pre-existing computer executable instructions in the second portion of memory only when writing the updated computer executable instructions from the first portion of the non-volatile memory to the third portion of non-volatile memory has completed.

17. The method of any preceding claim wherein the first portion of non-volatile memory is on a first non-volatile memory unit and the second portion of non-volatile memory is on a second distinct non-volatile memory unit.

18. The method of claim 17 wherein the first memory is one of an EEPROM or an SD card.

19. The method of any of claims 17 or 18 wherein the second memory portion is a flash ROM.

20. The method of any preceding claim wherein the computer executable instructions are firmware or boot code.

21. The method of any preceding claim wherein the communication link includes a wired connection that passes through a tether connecting the surface control computer to the ROV or to a Tether Management System, the tether containing power and data cables.

22. The method of any preceding claim wherein the communication link includes a wireless connection.

23. A functional module for an underwater remotely operated vehicle (ROV), the functional module comprising:

- one or more processors configured to access a first memory portion;

- a second memory portion that stores computer executable instructions, the one or more processors being further configured to execute computer executable instructions stored in the second memory portion in order to carry out one or more functions on the ROV;

the functional module being further configured to carry out the method of any of claims 1 to 22.

24. An underwater remotely operated vehicle (ROV) comprising one or more functional modules according to claim 23.

25. A system comprising a control computer and an ROV according to claim 24.

26. A computer program that when executed on a functional module causes it to carry out the method of any of claims 1 to 22.

27. A surface control computer that provides updates to one or more functional modules in an underwater ROV according to claim 23, the computer being configured to:

- identify and obtain the latest version of a set of computer executable instructions for the one or more modules;

- send updated computer executable instructions to the one or more functional modules of the ROV over a communication link;

- identify the one or more functional modules as unavailable;

- determine when the one or more modules have successfully updated the computer executable instructions; and

- mark each of the one or more modules as available when it is determined that they have successfully updated the computer executable instructions.

28. A method, functional module, ROV or surface control computer as hereinbefore described and with reference to the accompanying figures.

29. A method of updating one or more functional modules in an underwater remotely operated vehicle (ROV), the method comprising:

- receiving an update file at a functional module of the ROV from a surface control computer over a communication link;

- storing the update file in a first portion of non-volatile memory that is accessible by the processor of the functional module; and

- subsequently writing the update file from the first portion of the non-volatile memory to a second portion of non-volatile memory, wherein the functional module is configured such that one or more reconfigurable devices are updated as specified by the update file.

30. A method according to claim 29 wherein the one or more reconfigurable devices include one or more of a FPGA (Field Programmable Gate Arrays), a CPLD (Complex Programmable Logic Devices), and a PLC (Programmable Logic Controllers).

31. A functional module for an underwater remotely operated vehicle (ROV), the functional module being configured to carry out the method of any of claims 29 or 30.

32. An underwater remotely operated vehicle (ROV) comprising one or more functional modules according to claim 31.

33. A system comprising a control computer and an ROV according to claim 32.

34. A computer program that when executed on a functional module causes it to carry out the method of any of claims 29 or 30.

35. A surface control computer that provides updates to one or more functional modules in an underwater ROV according to claim 32, the computer being configured to:

- identify and obtain the latest version of an update file for the one or more modules;

- send an updated update file to the one or more functional modules of the ROV over a communication link;

- identify the one or more functional modules as unavailable;

- determine when the one or more modules have successfully updated; and

- mark each of the one or more modules as available when it is determined that they have successfully updated.

Intellectual Property Office

| | | | |
|---|---|---|---|
| **Application No:** | GB1610416.8 | **Examiner:** | Mr Andrew Stephens |
| **Claims searched:** | 1-35 | **Date of search:** | 18 July 2016 |

## Patents Act 1977: Search Report under Section 17

**Documents considered to be relevant:**

| Category | Relevant to claims | Identity of document and passage or figure of particular relevance |
|---|---|---|
| X | 1-35 | WO 2011/075139 A1 (HEWLETT-PACKARD DEVELOPMENT COMPANY); See whole document - particularly paragraphs [0015]-[0018], [0024], [0043]-[0046], [0060] & [0061] and Figs. 1-5 |
| X | 1-35 | US 2005/0160257 A1 (DELL PRODUCTS); See whole document - particularly paragraphs [0015]-[0019] and Figs. 3 & 4 |
| X | 1-35 | US 2004/0034861 A1 (BALLAI); See whole document - particularly paragraphs [0009], [0010] & [0018]-[0020] |
| X | 1-35 | WO 2013/126058 A1 (HEWLETT-PACKARD DEVELOPMENT COMPANY); See whole document - particularly paragraphs [0015]-[0017], [0025]-[0031] & [0035]-[0040] and Figs. 1-6 |
| X | 1-35 | US7480907 B1 (HEWLETT-PACKARD DEVELOPMENT COMPANY); See whole document - particularly col. 11, ln 44 - col. 12, ln 15 |

Categories:

| | | | |
|---|---|---|---|
| X | Document indicating lack of novelty or inventive step | A | Document indicating technological background and/or state of the art. |
| Y | Document indicating lack of inventive step if combined with one or more other documents of same category. | P | Document published on or after the declared priority date but before the filing date of this invention. |
| & | Member of the same patent family | E | Patent document published on or after, but with priority date earlier than, the filing date of this application. |

**Field of Search:**

Search of GB, EP, WO & US patent documents classified in the following areas of the UKC$^{X}$ :

Worldwide search of patent documents classified in the following areas of the IPC

G06F

The following online and other databases have been used in the preparation of this search report

WPI, EPODOC, TXTA, Internet

Intellectual Property Office

**International Classification:**

| Subclass | Subgroup | Valid From |
|---|---|---|
| G06F | 0009/445 | 01/01/2006 |
| B63G | 0008/00 | 01/01/2006 |