Amsterdam University of Applied Sciences

Rover Rescue System

Documentation

Author:	*Student Number:*
Christiaan van Arum	500778983
Raphaël Bunck	500774349
Nino van Galen	500790589
Martijn Vegter	500775388

January 14, 2019

Contents

1 Introduction

In the (near) future robots will be more and more part of our daily life. Even more than they are already part of society today. Industrial robots are nowadays very common, the use of drones by military and civilians triggers discussion and science is creating robots to take care of people who need help. Functioning more or less autonomous requires that such machines have to be robust, take their own decisions and operate in a safe way.

Thus we created a robot that helps to rescue people trapped in a collapsed building.

The user is able to control the robot using the combination of a controller, specifically a Steam Controller, and a moblie phone as display used by the Google Cardboard. Of course, being a robot, the robot itself wants to do as much as possible. The degree of autonomy of the robot has been focused on assisting the user as much as possible.

2 Installation Guide

This chapter gives a step by step tutorial to get the RSS up and running. After completing the installation, continue reading the usage guide.

2.1 Pre-installation

Before installing the Rover software, make sure you have the following software up and running:

- A RaspberryPi B2 (RPI) with newest version of Raspbian.
- An Active internet connection on the RPI.
- An open SSH, see setup below.

2.2 Headless setup procedure

- Step 1: Create an empty file. You can use Notepad on Windows or TextEdit to do so by creating a new file. Just name the file ssh. Save that empty file and dump it into boot partition (microSD).
- Step 2: Create another file name **wpa_supplicant.conf**. This time you need to write a few lines of text for this file. For this file, you need to use the **FULL VERSION** of wpa_supplicant.conf. Meaning you must have the 3 lines of data namely country, **ctrl_interface** and **update_config**.

```
}
country="Your country"
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
    ssid="your_real_wifi_ssid"
    scan_ssid=1
    psk="your_real_password"
    key_mgmt=WPA-PSK
}
```

2.3 Installation

To install Rover and its dependencies from your RPI shell prompt, use the following command:

```
$ wget -q https://git.io/fx0ko -O /tmp/rover && bash /tmp/rover
```

2.4 Manual Installation

Update your system package list:

```
$ sudo apt-get -update -y
```

Upgrade all you installed packages to their latest version:

```
$ sudo apt-get dist-upgrade -y
```

To download and install newest version of Node.js, use the following command:

```
$ curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
```

Now install it by running:

```
$ sudo apt-get install nodejs -y
```

To compile and install native addons from npm you also need to install build tools:

```
$ sudo apt-get install build-essential -y
```

To download the Rover code you need to install git:

```
$ sudo apt-get install git -y
```

Clone 'Scriptor' using git:

```
$ git clone https://github.com/RescueOnWheels/Scriptor.git
```

Clone 'Rover' using git:

```
$ git clone https://github.com/RescueOnWheels/Rover.git --recursive
```

Open the 'Scriptor' folder and start using the Rover:

```
$ cd Scriptor && ./start.sh all
```

3 Use Cases

Table 1: Connect to Rover

Name	Connect to rover
Brief	The user connects to the rover and enters the token
Actors	User, Rover and control device.
Assumption	The rover is installed and activated
Description	To use the rover the user must first connect to the rover.
Primary scenario	1. Connect all needed devices to the internet. 2. Enter IP in the application on the control device. 3. Enter Token on controller.
Exception	
Result	The user is connected to the rover and is able to operate it.

Table 2: Drive the Rover

Name	Drive Rover
Brief	The user connects to the rover and enters the token.
Actors	User, Rover and control device.
Assumption	The rover is installed, activated and connected.
Description	To use the rover the user must first connect to the rover.
Primary scenario	1. Press the button corresponding to your movement.
Exception	
Result	The rover moves in the chosen direction.

4 Definition of Ready

1. Must have clear description.
2. Must have a milestone.
3. Must have a estimate for the burndown chart.
4. Issues from the same type must be organised in epic's.
5. Relevant issues must be done before continuing.
6. Previous sprint issues must be done before new sprint issues. (Except icebox issues)
7. Issues can't be large and vague, should be split in multiple issues.

5 Definition of Done

5.1 Feature

1. DoD of each single User story, included in the Sprint are met
2. “To do‘s“ are completed
3. All unit tests passing
4. Product backlog updated
5. Project deployed on the test environment identical to production platform
6. Tests on devices/browsers listed in documentation passed
7. Tests of backward compatibility passed
8. The performance tests passed
9. All bugs fixed
10. Sprint marked as ready for the production deployment by the Product Owner

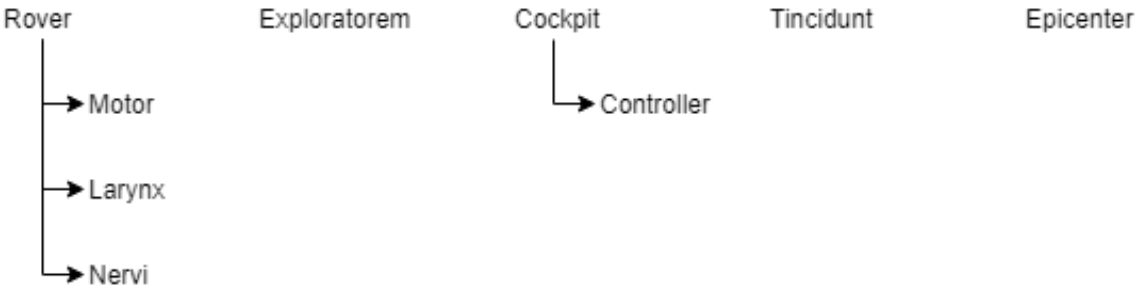
5.2 Sprint

1. Code Complete
2. Environments are prepared for release
3. All unit & functional tests are green
4. All the acceptance criterias are met
5. QA is done & all issues resolved
6. All “To Do“ annotations must have been resolved
7. OK from the team: UX designer, developer, software architect, project manager, product owner, QA, etc.
8. Check that no unintegrated work in progress has been left in any development or staging environment.
9. Check that TDD and continuous integration is verified and working

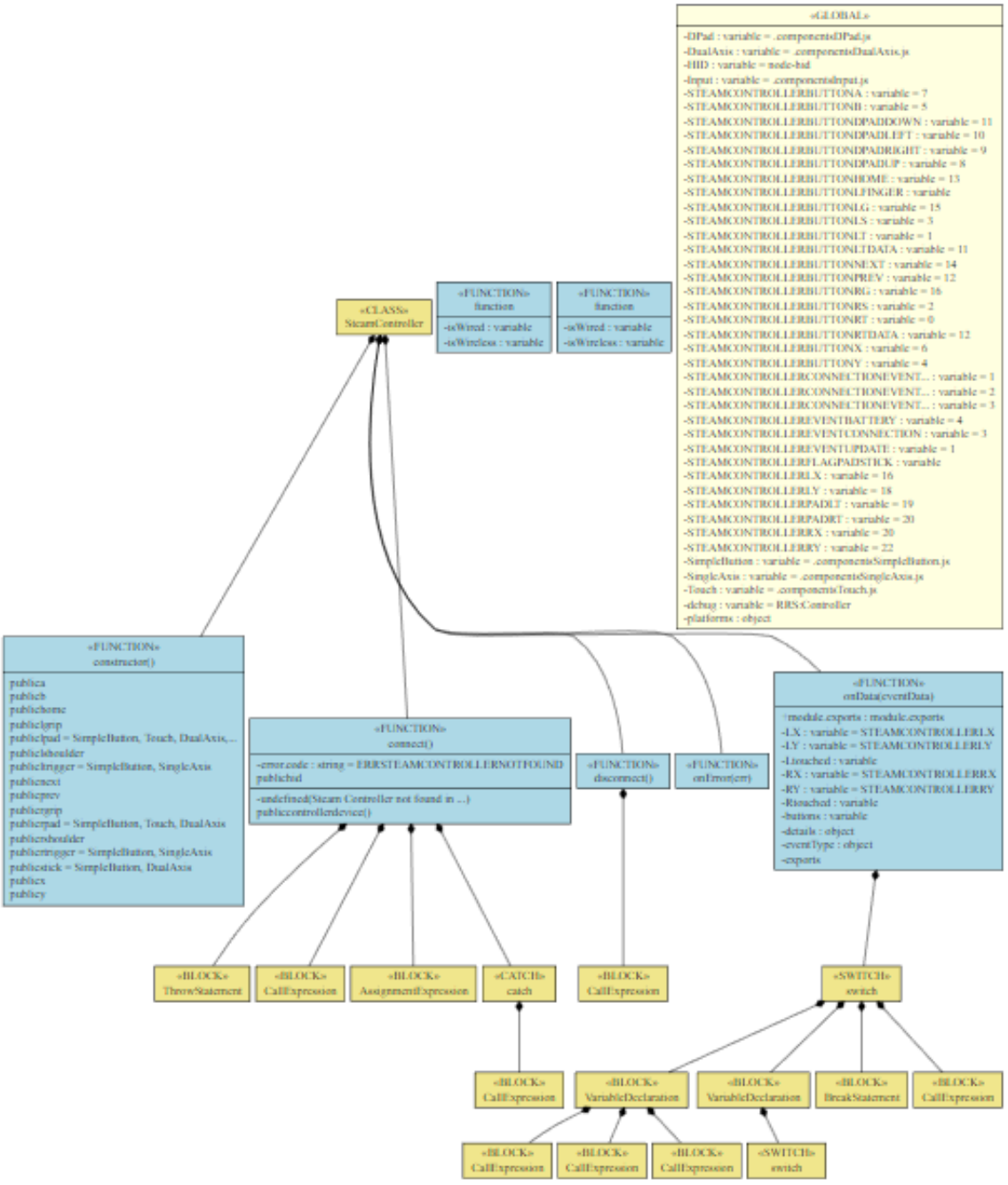
6 Class Diagrams

The class diagrams are a visual representation on how the RRS has been structured. The project overview given below shows which modules are submodules from the overheading modules. In the UML diagrams below there has been given a precise and complete overview of the class diagrams per module. These diagrams have been made using a generating tool in a command line interface.

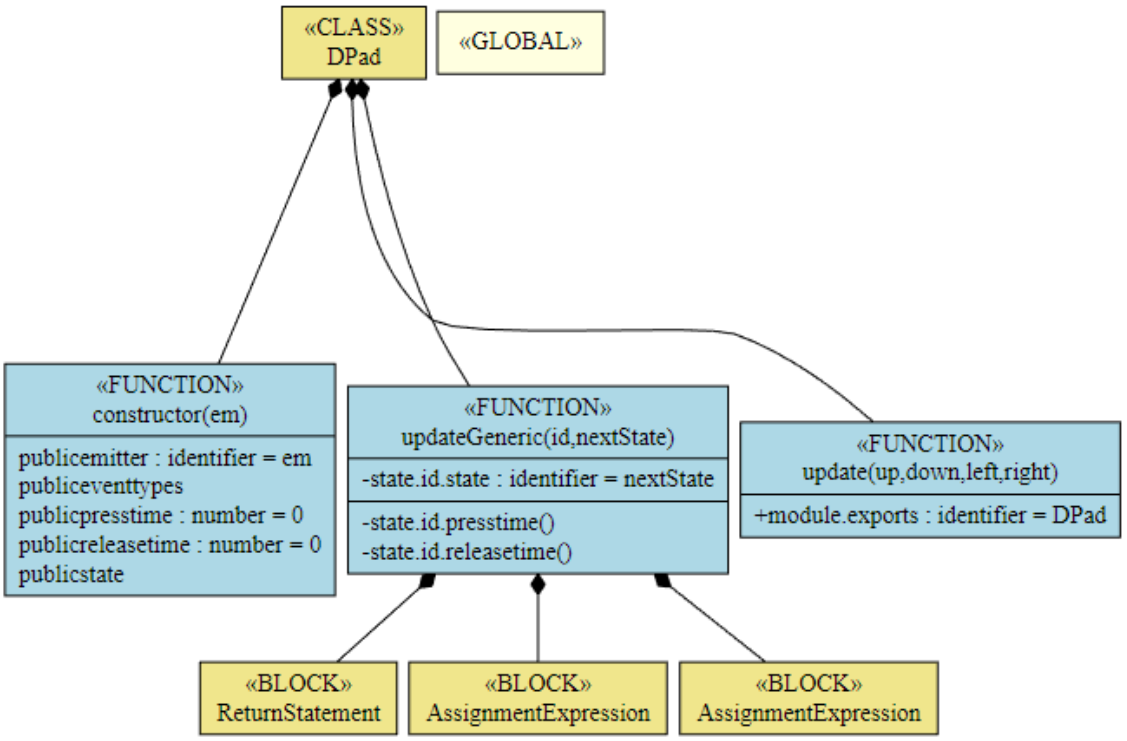
6.1 Project Overview



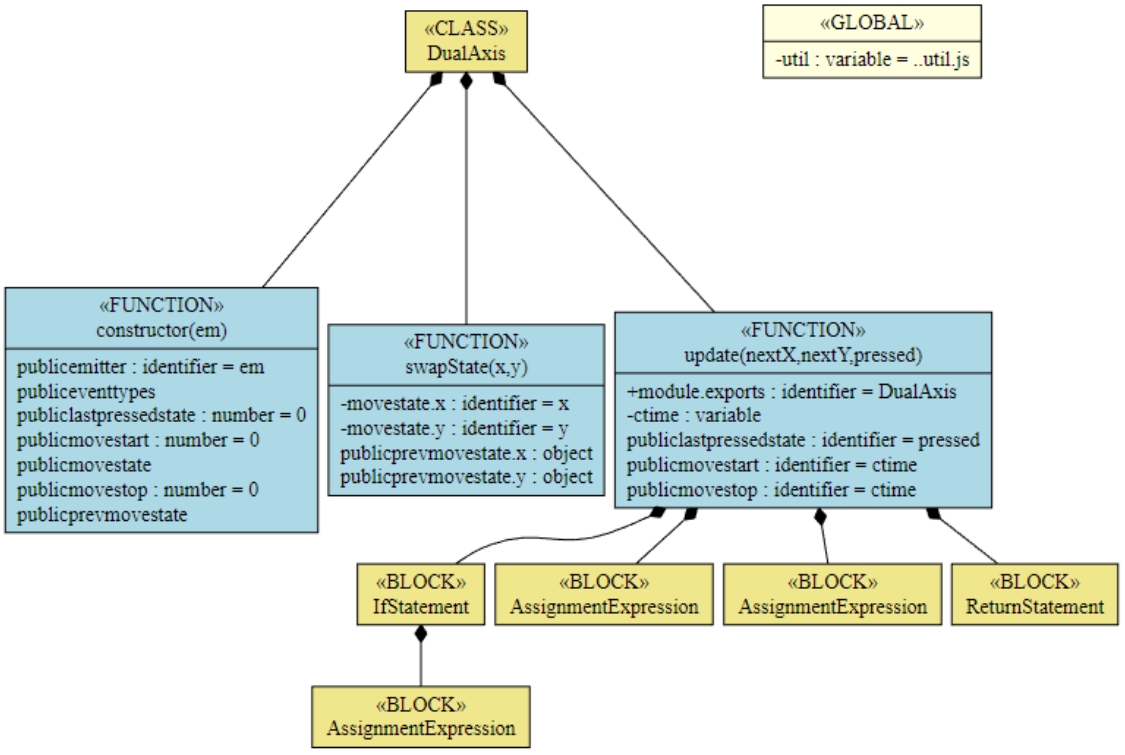
6.2 Controller - Steam Controller



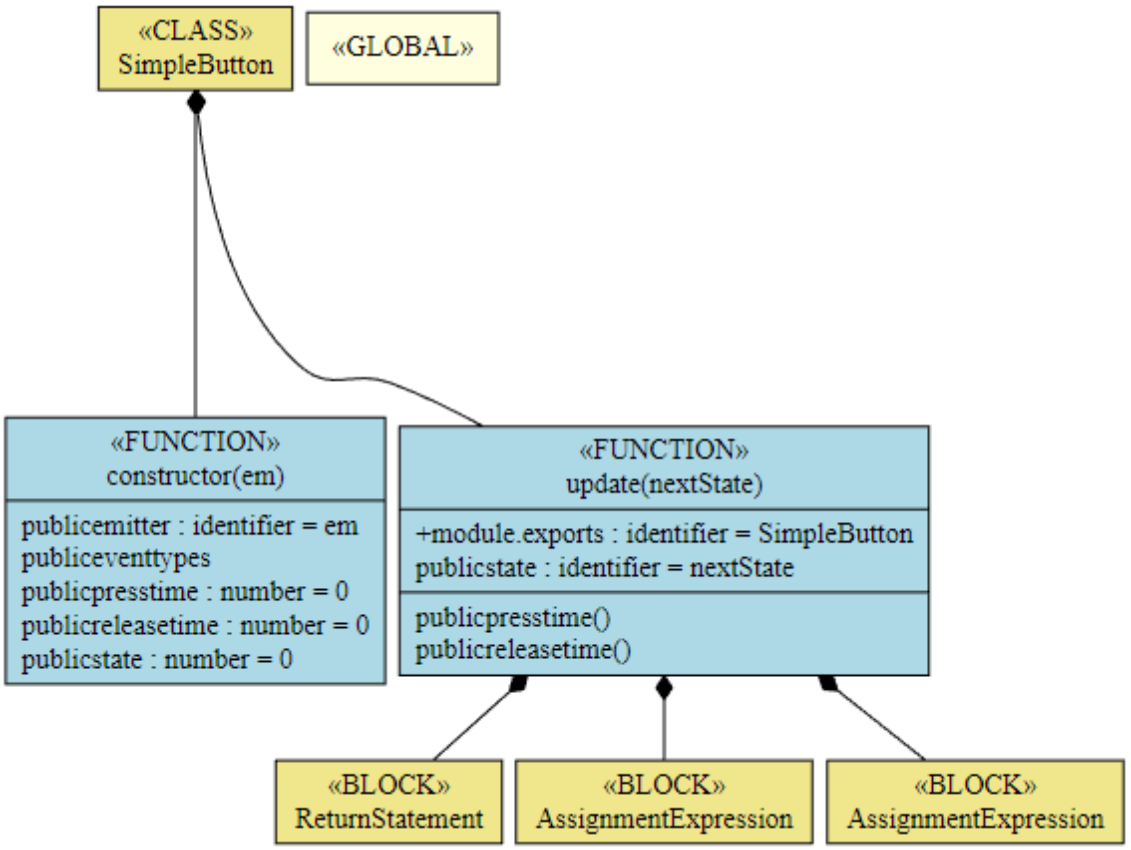
6.3 Controller - DPad



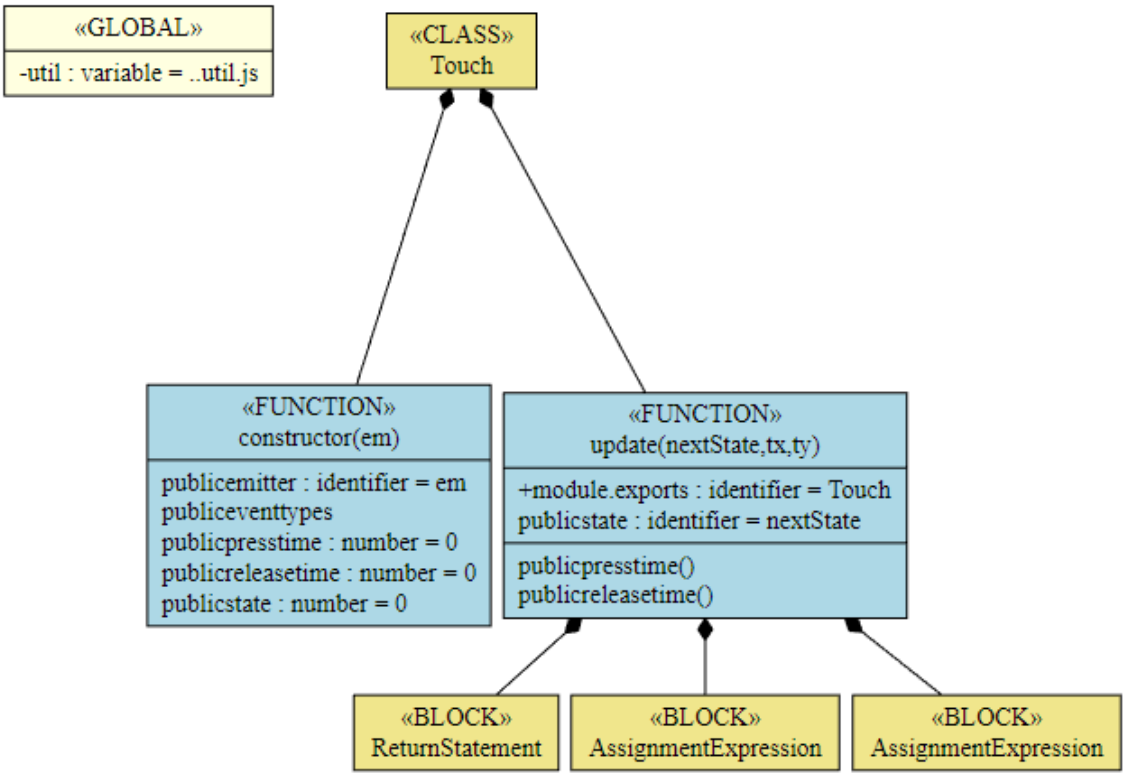
6.4 Controller - Dual Axis



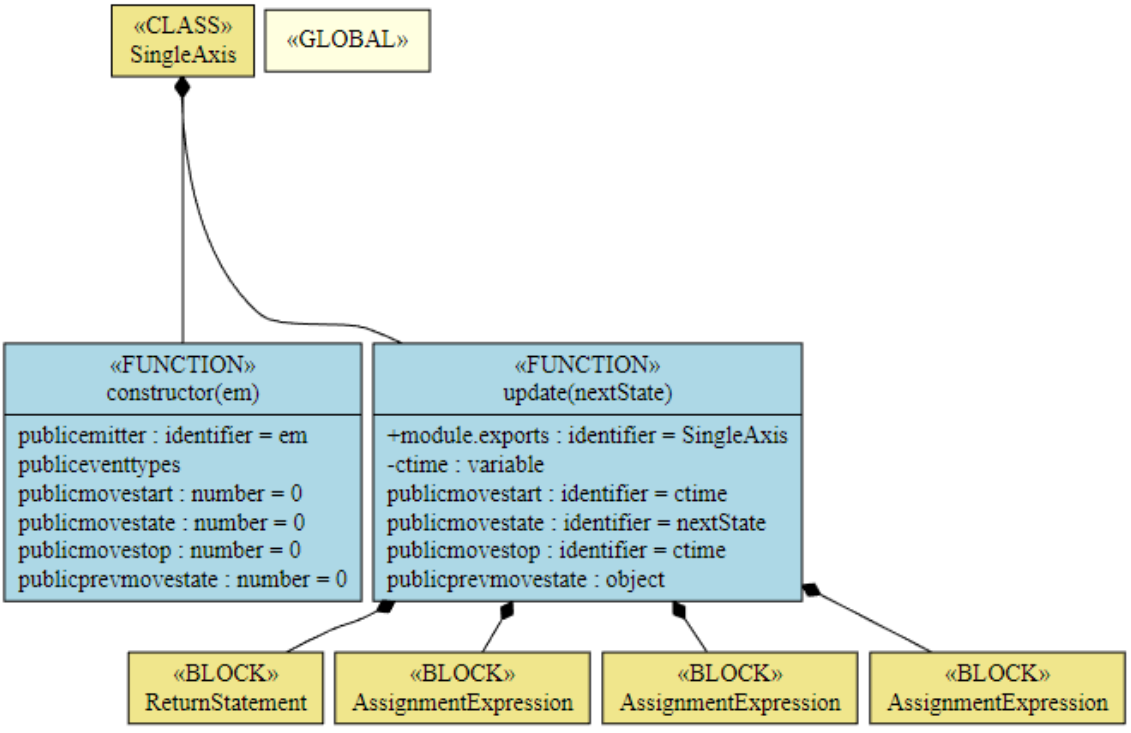
6.5 Controller - Simple Button



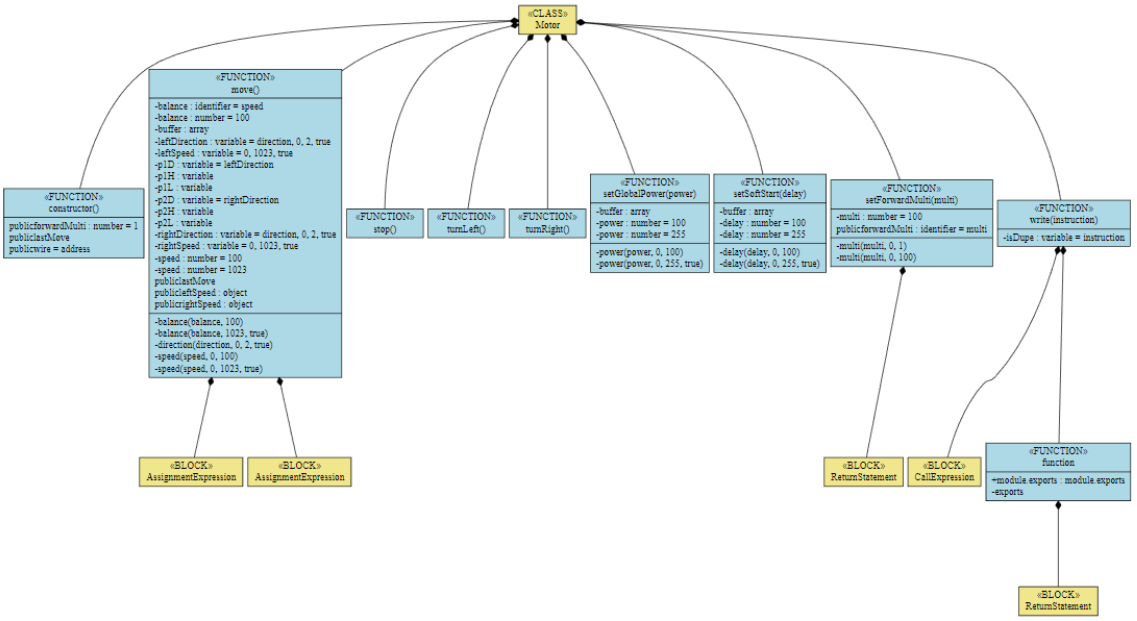
6.6 Controller - Touch



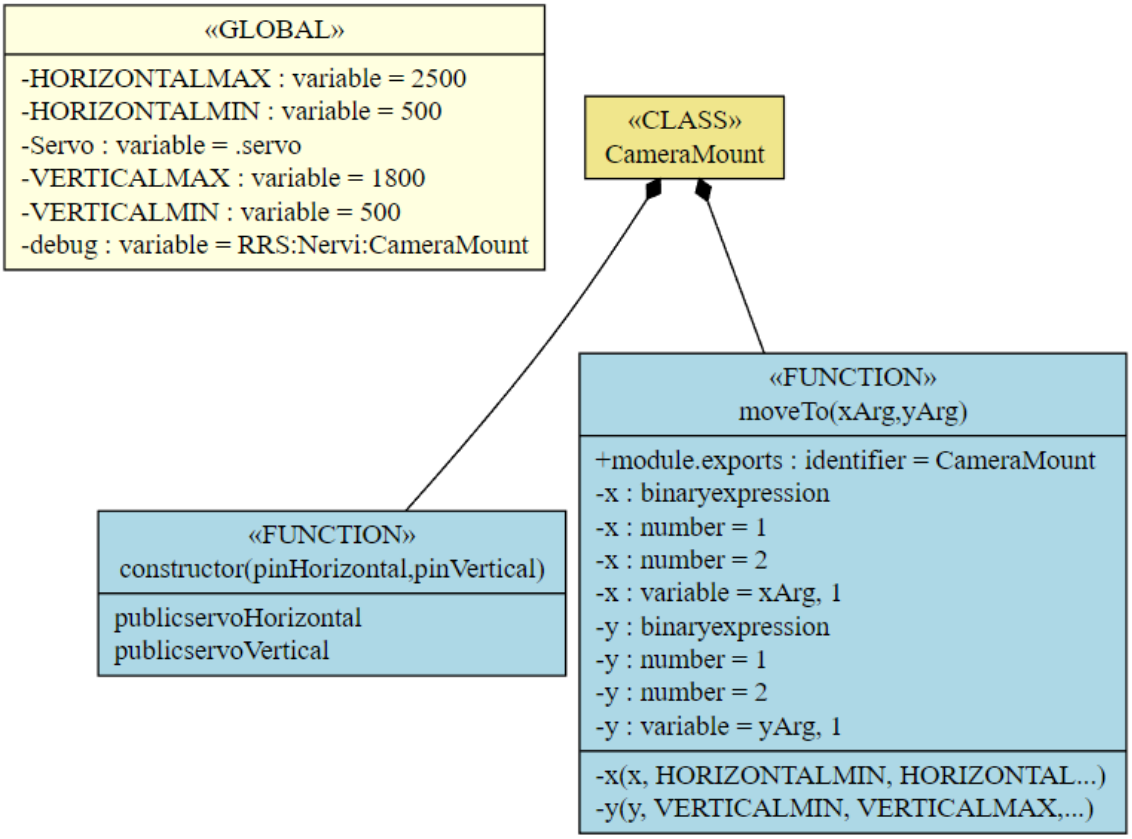
6.7 Controller - Single Axis



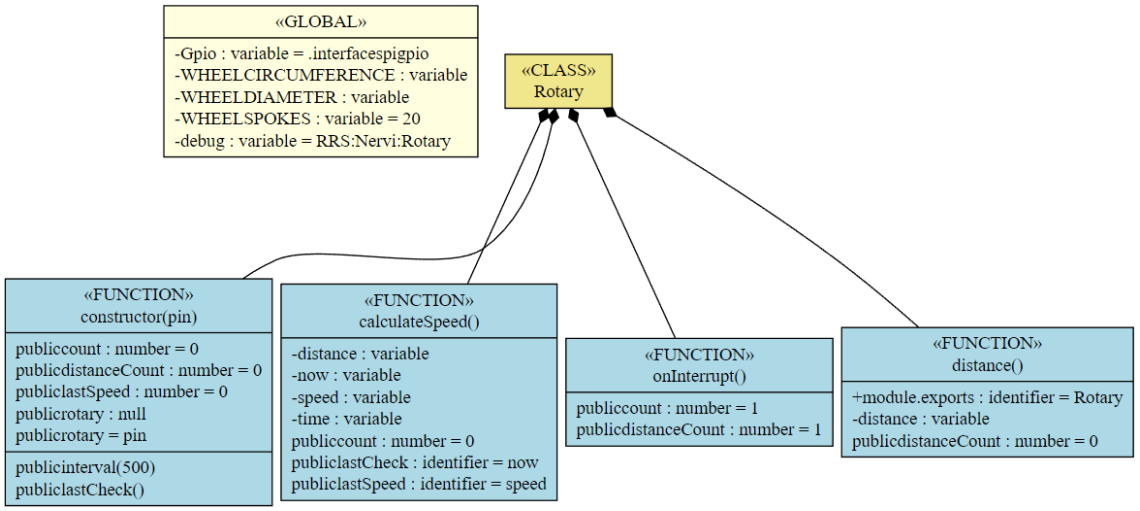
6.8 Motor


«CLASS»
Motor
«FUNCTION»
constructor()
publicforwardMulti : number = 1
publiclastMove
publicwire = address
«FUNCTION»
move()
-balance : identifier = speed
-balance : number = 100
-buffer : array
-leftDirection : variable = direction, 0, 2, true
-leftSpeed : variable = 0, 1023, true
-p1D : variable = leftDirection
-p1H : variable
-p1L : variable
-p2D : variable = rightDirection
-p2H : variable
-p2L : variable
-rightDirection : variable = direction, 0, 2, true
-rightSpeed : variable = 0, 1023, true
-speed : number = 100
-speed : number = 1023
publiclastMove
publicleftSpeed : object
publicrightSpeed : object
-balance(balance, 100)
-balance(balance, 1023, true)
-direction(direction, 0, 2, true)
-speed(speed, 0, 100)
-speed(speed, 0, 1023, true)
«FUNCTION»
stop()
«FUNCTION»
turnLeft()
«FUNCTION»
turnRight()
«FUNCTION»
setGlobalPower(power)
-buffer : array
-power : number = 100
-power : number = 255
-power(power, 0, 100)
-power(power, 0, 255, true)
«FUNCTION»
setSoftStart(delay)
-buffer : array
-delay : number = 100
-delay : number = 255
-delay(delay, 0, 100)
-delay(delay, 0, 255, true)
«FUNCTION»
setForwardMulti(multi)
-multi : number = 100
publicforwardMulti : identifier = multi
-multi(multi, 0, 1)
-multi(multi, 0, 100)
«FUNCTION»
write(instruction)
-isDupe : variable = instruction
«BLOCK»
AssignmentExpression
«BLOCK»
AssignmentExpression
«BLOCK»
ReturnStatement
«BLOCK»
CallExpression
«FUNCTION»
function
+module.exports : module.exports
-exports
«BLOCK»
ReturnStatement

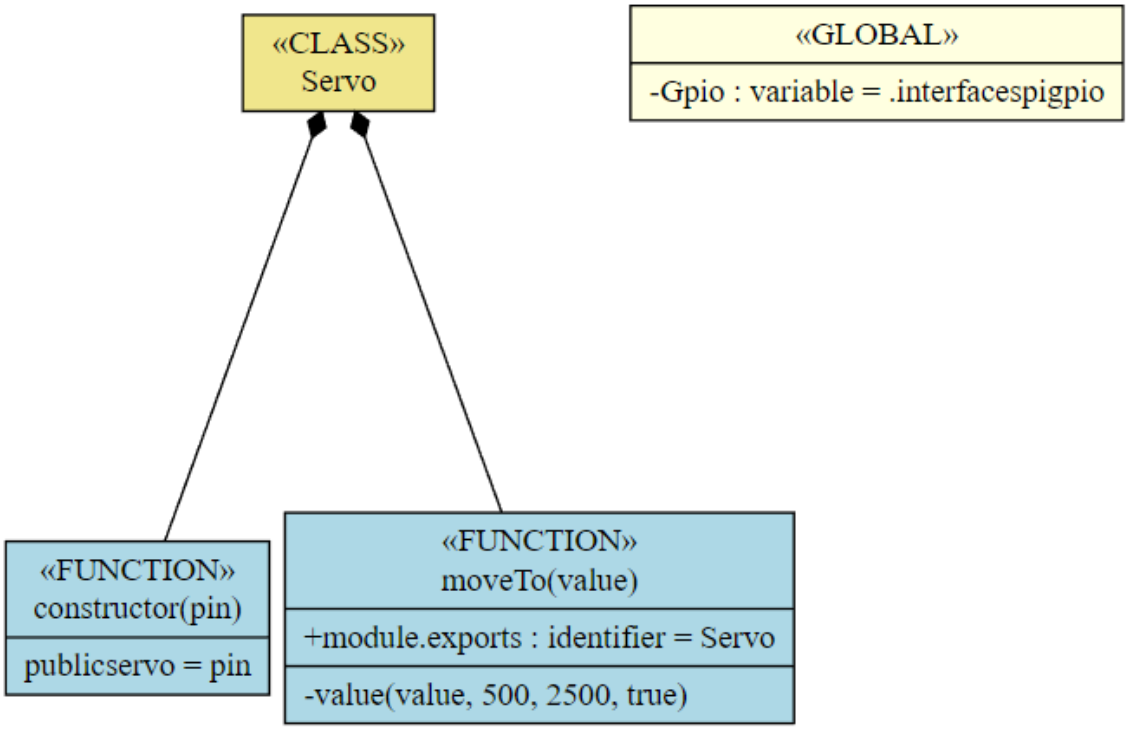
6.9 Nervi - Camera Mount



6.10 Nervi - Rotary Encoder



6.11 Nervi - Servo



6.12 Nervi - Ultrasonic