**Husqvarna Project - Hedgehog**

Introduction

The goal of this project is to convert ROS program which manage currently the lawnmover to Opendlv software. There will be a Rasperry Pi insert in the lawnmover to control it. Besides the Rasperry Pi is almost ready. It is in the project locker : the last version of Raspbian and Docker is installed. We want 2 microservices for this device : one for the control/joystick and one to analyze the command and make the lawnmover moving. The microservice for the joystick will be « device - gamepad ».

**There were different steps and reconsideration in this project. Here is a little chronology :**

First I try to analyze all ROS program/automower\_safe file which run the actual Hedgehog lawnmover. The purpose was to make a mapping of the messages/commands to be able to adapt it on opendlv. Then I try to find which file from Kiwi can be re-used for the lawnmover : Opendlv-sim-motor or/and opendlv-device-prugw seems first to be the good one.

* Opendlv-sim-motor

It received data from opendlv-view with « Sender-stamp » parameter in the file sim-motor.cpp. As the single track model, it commands the vehicle according parameters like angle,torque … The singletrack model needs some values as mass of the vehicle, lenght.

* Opendlv-device-prugw

With programs like Motor.cpp/.h , the lawnmover get & set power to the servo or esc motor according the value of « m\_power ». The program prugw.cpp analyzes sender-stamp() message from data ::envelope.

PwmMotors :: setpower() give power to the good chanel. Besides the channel (which servo motor) is choosen in PwmMotors :: pwmMotors.

**After analyzing and understanding these programs, we can see that there are too much difference with the lawnmover running. That’s why we cannot use them. The lawnmover don’t use escp and servo motor for instance. There are just two motors, one for each wheel.**

Then I used **seacat file** to create an opendlv-gw-hedgehog file. It will contain all the programs for the correct functioning of the lawnmover.

# Command of the lawnmover / Communication

## Study of ROS command interface

The communication with the automower happened in **auto\_mower\_safe.cpp** with **AutomowerSafe::sendMessage()** function. The ROS module am\_driver\_safe sends and receives data via a serial port (can also be via USB, with same protocol).

Commands are converted from text form to hcp-protocol internally in the **automower\_safe class**.

This happens in the **sendMessage () method**, which also sends the command to the serialport.

am\_driver\_safe also has a rule algorithm that controls the two wheel-motors of the robot so that the desired speed and direction are obtained.

In order for AutomowerSafe::sendMessage to work, you must also pick up the hcp-library that consists of the c-code routines located in the folders am\_driver\_safe/src/hcp and am\_driver\_safe/include/hcp.

All available commands that can be sent to the automower are in the **am\_driver\_safe/config/automower\_hrp.json** file.It is this file that the hcp-library uses to translate commands in text-form-strings to data sent over the serial port. This file needs certainly to be add into the new environment.

hrp\_teleop.py :

* It sends messages to cmd\_vel

Am-driver\_safe :

* Sends messages to the motherboard of the lawnmover
* Listens to cmd\_vel

*am\_driver\_safe/config/automower\_hrp.json file command (= all commands that lawnmover understand🡪what can be send or receive)*

sendMessage() function to recover

*automower\_safe = use theses commands to do some actions on the lawnmover (move it with wheelpower for instance)or understand data from the lawnmover*

*hrp\_teleop = send messages to cmd\_vel (it’s in sort the joystick)*

To understand we can take the example of Battery data :

In hrp.json 🡪 GetBatteryData from the family RealTimeData has a variable batavoltage.

In automower\_safe, we use RealTimeData.GetBatteryData() with the new variable batterystatus.batteryAvoltage=batAvoltage

But in hrp\_teleop, we just have the function :

def callback\_battery\_status(self,data) :

self.battAvolt=data.batteryAvoltage/1000

***To conclude, for the battery\_status, the mapping is : automower-hrp.json 🡪 automower\_safe.cpp 🡪 hrp\_teleop.py***

**According to the kind of data, there is a single or double way of exchange between the three files. It’s due to « subscriber » or « publisher » option. For the management of speed dat, it is in the other way.**

## Inventory of the commands understandable by the lawnmover

Here are all messages understandable by the lawnmover (in automower-hrp.json).

* Methods DeviceInformation.GetdeviceIdentification

Use in **automower\_safe = bool AutomowerSafe ::initAutomowerBoard** to move and init the mower

* Wheels.GetRotationCounter

Use in **Automower ::getEncoderData()** to have the rotation speed and so adapt with motorfeedbackdiffdrive.ticks and .stamp for the time. It gives especially left or right pulses if it’s necessary. According to the index 0 or 1, we can choose right or left wheel.

*Where is defind Left/rightPulses ?*

* RealTimeData.GetWheelMotorData

Use in **Automower ::getWheelData()** to receive the data and adapt with motorfeedbackdiffdrive through omega, current, power

* MowerApp.GetState

To get state of the lawnmover in **AutomowerSafe ::getStateData().**

Use all tIMowerState\_Mower mode like « IMOWERAPP\_STATE\_OFF  or PAUSE »

* SystemSettings.GetLoopDetection

Use in **Automower ::getLoopDetecton()**. It gives messages like DISC\_ON, LOOP\_ON,PARKED

* SafetySupervisor.GetStatus

Use in **Automower ::getStatus** . It gives messages like CHARGING,COLLISION, USER STOP, LIFTED.

* Charger .IsChargingPowerConnected

Used in **Automower ::getChargingPowerConnected()** . It gives messages like IN\_CS 🡪 check if lawnmover connecting to the station

* CurrentSattus.GetStatusKeepAlive

Used in Automower **::getStatuskeepAlive**. It’s to prevent the lawnmover to go to sleep mode.

* RealTimeData.GetSensorData()

Used in **AutomowerSafe ::getSensorStatus** 🡪 this methods use getLoopdtection, getStatus, getchargingPowerConnected, getstatuskeepalive.

* Wheels.poweroff()

Used in **Automower ::stopWheels**() to clear PIDS and power

* HardwareControl.WheelMotorsPower

Used in **Automowersafe ::sendwheelPower**( double power\_left, double power\_right) to send power to the wheels right or left or both of them.

* MowerApp.Pause

Used in **AutoMowerSafe ::pauseMower()**

* MowerApp.StartTrigger

Used in **AutomowerSafe ::startmower()**

* MowerApp.SetMode

Used in **AutomowerSafe ::setAutoMode()** and AutomowerSafe **::setParkMode()**with respectively IMOWERAPP\_MODE\_AUTO and IMOWRAPP\_MODE\_HOME

* SystemSettings.SetLoopdetection

Used in **AutomowerSafe ::loopdetectionhandling()**

* BladeMotor.Brake

Used in **AutomowerSafe ::cutDiscOff()** . Besides there is also a BladeMotor.Run msgType. We don’t need the other functions because the blade has been removed.

Conclusion

In yellow are all the messages which seems really necessary to make move the lawnmover. But to just make forward the device, **first** we will only use **HardwareControl.WheelMotorsPower** message and maybe DeviceInformation.GetdeviceIdentification & MowerApp.StartTrigger

## Time Management

We can adjust which data is read at what frequency by adjusting the parameters in the file **am\_driver\_safe/launch/automower\_hrp.launch. But these frequencies manage the sensors principally, so is it useful in our case ?**

The function bool AutomowerSafe::update(ros::Duration dt) manages the time

# Study of hrp\_teleop

On hrp\_teleop, some important variables are initialized :

* Self.speed=np.array([0,3,1.0])
* Self.command=np.array([0,0])
* Self.inc\_ratio=0,1

The publishers are also created :

* self.pub\_twist = rospy.Publisher('/cmd\_vel', Twist, queue\_size=1)
* self.pub\_mode = rospy.Publisher('/cmd\_mode', UInt16, queue\_size=1)

## 2.1 Speed data

def update()

self.pub\_twist.publish(twist)

**(publish speed and command information)**

**Twist parameter expresses velocity in free space broken into its linear and angular parts.**

def process\_key(self,ch)

ch = key

self.speed=

self.command=

* **According to value of key, do an action and save it**
* **It sends twist messages thanks to update() function at a certain frequency rate**

def get\_key(self)

return key

def run(self)

r=rospy.Rate(self.update\_rate) #10Hz

ch=self.get\_key()

self.update()

* **Recover keyboard**
* **key**

**Command :**

Q

A

E

W

D

W

C

Z

X

In the calculation of the speed, we can only consider the command cmd\_bindigns to simplify the new device. We will not adjust the speed.

## Time update and rospy information

In hrp\_teleop, the time is update through rospy function.

* Rospy

rospy is a pure Python client library for ROS.There are some functions proper to ROS like publisher and subscribers.

<http://wiki.ros.org/rospy/Overview/Publishers%20and%20Subscribers>

* Time

<http://wiki.ros.org/rospy/Overview/Time>

ROS has the ability to setup a simulated [Clock](http://wiki.ros.org/Clock) for nodes. Besides rospy provides a rospy.Rate convenience class which makes a best effort at maintaining a particular rate for a loop.

But in fact, the time will maybe too complicated to manage in a first time. We just want to make move in an esay way the device. So this quston will be studied in the future.

# Study of automower\_safe

## Speed data management

So now the speed request is recovered by automower\_safe.cpp in **Automower::velocityCallback ()** function via geometry ::twist message. Then new variables are created : wanted\_lv & wanted\_rv thanks to linear and angular data.

**Automower::velocityCallback (**geometry ::twist message**)**

Wanted\_lv=

Wanted\_rv=

**AutomowerSafe ::RegulateVelocity()**

power\_l=f(current\_lv,wanted\_lv)

power\_r=f(current\_rv,wanted\_rv)

sendWheelPower(power\_l,power\_r)

**Automower::getWheelData ()**

current\_lv=

current\_rv=

It uses RealTimeData.GetwheelMotorData() msg from hrp.json

It uses leftWheelPid.Update and rightWheelPid.Update to make the link

**AutomowerSafe::sendWheelPower(double power\_left, double power\_right)**

Send to lawnmover through automower ::sendwheelpower via leftwheelmotorpower variable (hrp.jason)

**Lawnmover**

So the management of speed date uses PidRegulator class and Update method. RegulatePid(current\_vel, wanted\_vel) method from Automower class is also used. We can notice that there are classes creator with *PidRegulator leftwheelPid* and *PidRegulator rightWheelPid* in Automower class.

In fact, there is a conversion from velocity data (recover from teleop) to power thanks to PID calculation. This conversion is necessary because the lawnmover just understand power information. The velocity is expressed in m/s but unfortunately we don’t know the unit of power (>100 or <-100). Indeed we don’t have the unit of the proportional or integral coefficient in the calculation.

That’s why at the beginning I try to create a program which make the conversion but it was a little too complicated. **We will only use power in the next step to make move the lawnmover.**

## Other function/command which seems important

So now we will just quote all the primordial function to make move the lawnmover in an easy way.

* **Automowersafe ::Automowersafe()** As a constructor, it initializes the parameters and recover data from hrp\_teleop via cmd\_vel buses
  + velocity\_sub=nb.suscribe(cmd\_vel)
  + Time management : ros ::duration
  + Initialize HCP library, internal state and json file
* **AutomowerSafe ::sendMessage()**to send and receive data to the automower
* **AutomowerSafe ::loadJsonModel() :** it’s used in automowersafe ::automowersafe() to load json file
* **AutomowerSafe ::setup()**
* **AutomowerSafe ::velocityCallback(geometry\_msgs ::twist)** recover speed instruction
* **AutomowerSafe ::modeCallback()** It uses cmd\_mode (hrp\_teleop) to change the mode of the automower. We have to code all these functions to directly make them true :
  + **setManualMode()**

**ROS-INFO (« Random mode »)**

**eventQueue -> raiseevent(« /Manual »)**

It has to be changed

* + **system(« shutdown »)**
* **bool AutomowerSafe ::initAutomowerBoard()** . It’s to define the type of lawnmover accoding to GetdeviceIdentification data (json). It enable to recognize the lawnmover model. It uses DeviceInformation.GetDeviceIdentification to communicate with the lawnmover.
* **AutomowerSafe ::GetwheelData()** Participate in the convertion of velocity into power
* **AutomowerSafe ::stopWheels()**
* **AutomowerSafe ::regulateVelocity()** Enable to send power to left and right wheel through pid regulation
* **AutomowerSafe::sendWheelpower(double power\_left,power\_right)** to convert speed data from controller/joystick to power (don’t know what is the unit of 100 ??)
* **AutomowerSafe ::update(ros ::duration) through** a time management, it updates function as wheelgetdata
* **AutomowerSafe ::startMower()** Uses MowerApp.Starttrigger msg (json)
* **AutomowerSafe ::setAutoMode()** UsesMowerApp.SetMode(modeOfOperation:%d)",IMOWERAPP\_MODE\_AUTO)

## Time management

Many ros ::timre function which manage the time and AutomowerSafe ::update(ros ::Duration dt) which define the state according the time response of the lawnmover or manage the sensor when the lawnmover is connected = getWheelData is maybe important for us in this function and also regulateVelocity()

There are no special link between logic\_lawnmover and hrp\_teleop, we can use the example of motorFeedbackDiffdrive :

Motorfeedbackdiffdrive msg recovers the data coming from Wheels.GetRotationCounter (hrp.json) especially left or right pulses in AutomowerSafe ::getEncoderData() and recover omega and power in Automower ::getWheelData(). Then the values are published in AutomowerSafe ::update(ros Duration).

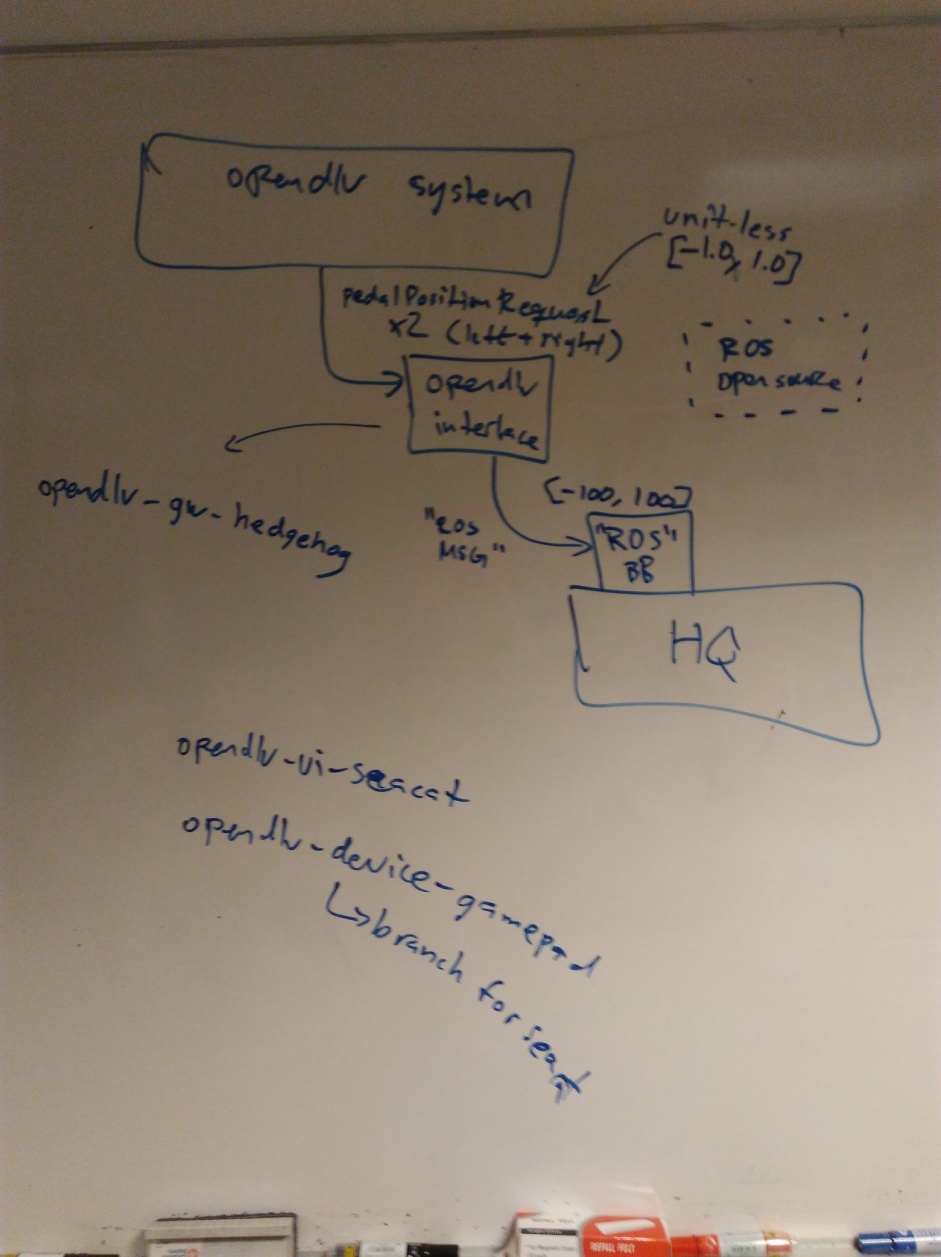
## Issues with ROS functions

There is a specific class ROS with some functions like publishr, subscriber and eventQueue.

<http://docs.ros.org/hydro/api/decision_making/html/classdecision__making_1_1RosEventQueue.html>

# Transition to Opendlv

## 4.1 Opendlv interface



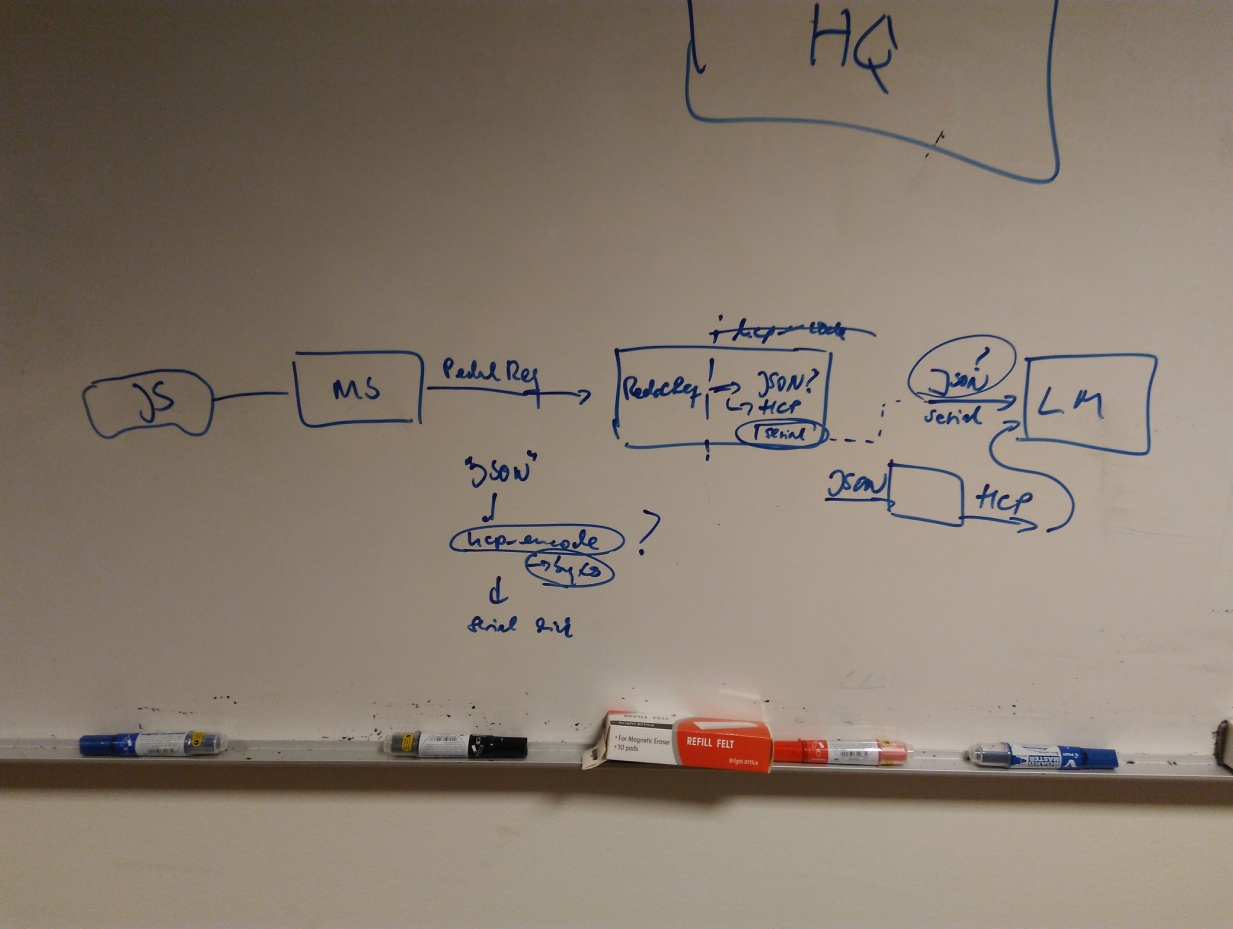
So now we’ll try to begin the new code (in the file hedgehog of Github, in Arnaud fork). As we said previously, we will only use power data. So pedalPositionRequest (left & right) messsage from Standard message will be used. It’s unitless but we will treat the data in order to make the similar comparison as ROS structure. We will use opendlv-device-gamepad and especially the seacat branch tto establish the new code « opendlv-gw-hedgehog ».

## 5.2 What happens between json message and the lawnmover ?

To make the new code, we have to understand more precisely what happens between json message and the lawnmover and especially all hcp command/messages in automower ::sendmessage().Indeed it’s the function which make the link between the command and the lawnmover. So we need a clear mapping and know if we have to re-use hcp message as the picture beneath shows (arrow n°2)

**2**

**1**



The function sendmessage() enables to communicate (send and receive) with the lawnmover through json messsage. It returns true or false according the success of the communciation. Inside the function hcp-encode encodes messages to serialport. It used especially a buffer to copy data into the serialport. And these serialport are essential for the device functioning.

* Hcp\_Encode(hcp\_tState\* pState, hcp\_Size\_t CodecId, hcp\_cszStr Command, hcp\_Uint8\* pDestination, hcp\_Uint32 MaxLength)

It encodes a request into a byte-array. The output pdestination is a buffer. On success, the function returns a value greater to zero which indicates how many bytes that were written to the pdestination. A value less than zero indicates an error.

* So now the question is : where the source code of the hcp\_encode message that we need for the low-level communication is ?

I don’t think that hcp\_encode is used somewhere else for the communication with the lawnmover, but it has to be verify.

**Moreover if the hcp\_encode expects JSON to produce the low-level commands for the serial line,**

**we might be able to skip the JSON part entirely in the new device by incorporating the hcp\_encode method correctly.**

.

Conclusion

To conclude, I’ll try to make a list of the next step to advance in this project :

* Find and verify where hcp-encode is used in the source code 🡪 in order to know the structure and especially if we can skip JSON part in the new device (remove the loop which recover and send message to the lawnmover via sendmessage and by using json message)
* Verify the code opendlv-gw-hedgehog 🡪 correct link between right and left variable from the gamepad and the PedalPostioRequest which recover the data.
* New code to make a link between PedalPositionRequest (left & right value) and hcp\_encode/serialport from the lawnmover (**arrow 2)** 🡪 This second enable the lawnmover to understand the command.
* Time management, especially in opendlv-gw-hedgehog « cluon ::time » 🡪 verify/change , maybe is it useless in a first time ? Otherwise it has to be present in the second part
* Include all these new programs in the Rasperry Pi which is already ready in the project locker 🡪 Raspian & docker is installed on it