yRobot-FreeRTOS.xml *rev1-3* *fev-2021*

Description des Tasks et Queues de :
WS_Cpp\yRobot

```
yTask.h:
typedef enum  {
  TkNone = 0,
  TkInit,
  TkDefault,
  TkMoteurD,
  TkMoteurG,
  TkTrain,
  TkProcess,
  TkCheckVR,
  TkVTaffiche,
  TkAll,
} yTkOrder;
```

Les tâches sont démarrées ds l'ordre 'yTkOrder'

derniere modif en ROUGE

task robot
tk train
tk_moteur

VRx = [-90, +90] angle de direction

tk_Init

S'exécuté une fois!!
Démarre les taches ds ordre 'yTkOrder'
Auto delete

tk_Default

Some debug tests

EXTI15_10_IRQHandler(BP1)

Sur Interrupt
Envoie MsgX ds queue

.Topic=BP1

EXTI2_IRQHandler(SWxy)

Sur Interrupt
Envoie MsgX ds queue

.Topic=SWxy

tk_CheckVR

Toutes le temps
Analyse le joystrick VRx, VRy,SWxy
Envoie MsgX ds queue

.Topic=VRx, PayLoadF=VRx.PV
.
Topic=VRy, PayLoadF=VRy.PV

qEvents
MsgX

tk_Process

* recuoérer un event ds la queue
* afficher le contenu de l'event (via queue de tk_VTaffiche)
* swirch Topic
  case BP1
  case SWxy
  case Vrx
  case VRy
  case kbd

yMenu.c/
yMenuGetTouche

aRxBuffer[0]

A

N

B

N

.Topic=kbd, PayLoadI=0U

.Topic=kbd, PayLoadI=1U

MsgX

```
yTask.h:
typedef struct {
  uint8_t Topic;
  float   PayLoadF;
  uint8_t PayLoadI;
} yEvent_t;
```

```
yTask.h:
typedef enum {
  TopicNone  = 0,
  BP1   = 1,
  SWxy  = 2,
  kbd   = 3,
  Mar   = 4,
  Arr   = 5,
  VRx   = 11,
  VRy   = 12,
  DIR   = 13,
  SP    = 14,
  MotD  = 21,
  MotG  = 22,
  Ro    = 31,
} yTopic;
```

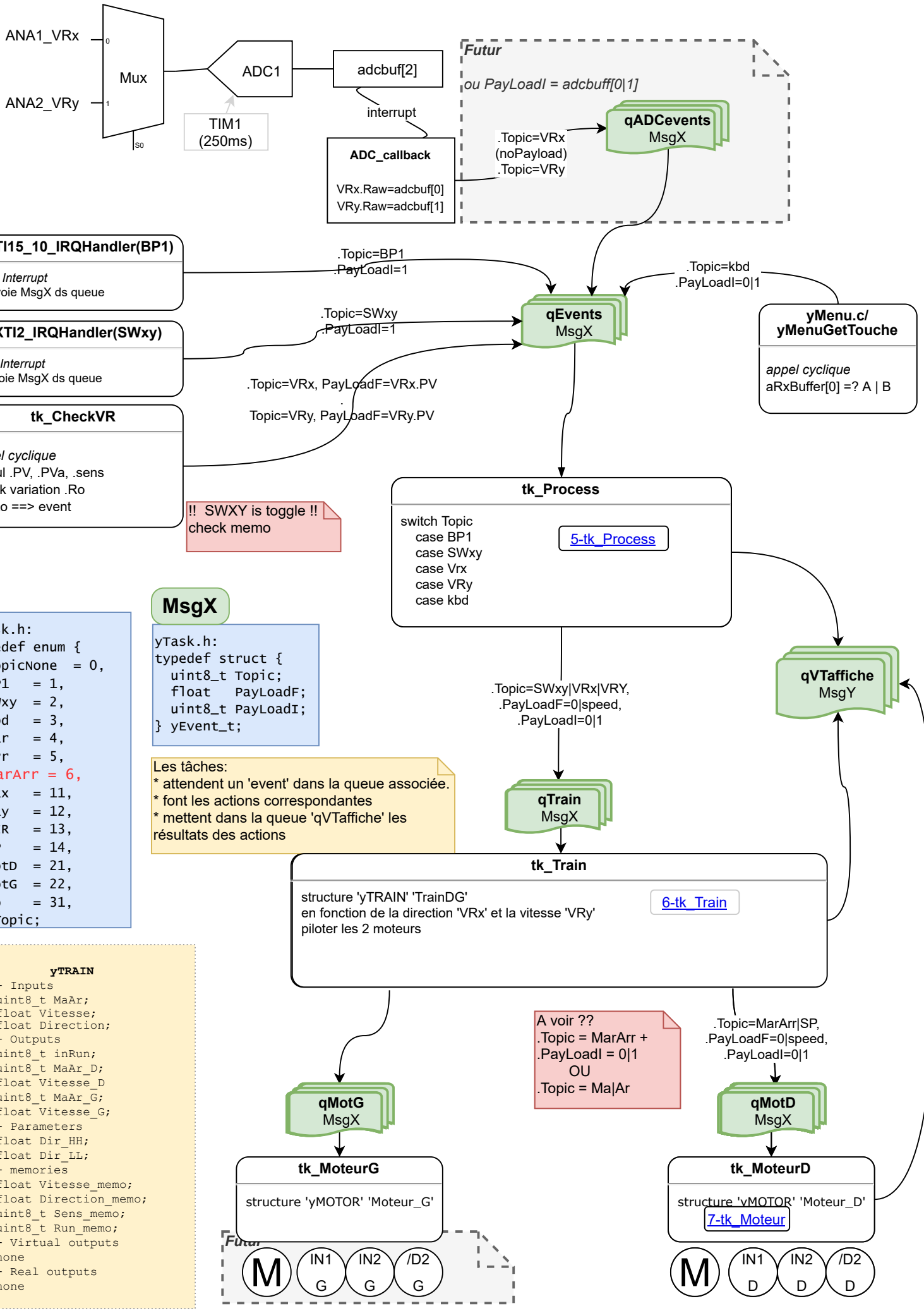
```
yTask.h:
typedef enum {
  DestNone  = 50,
  tkProcess = 51,
  VTaffiche = 52,

} yDestination;
```

```
yTask.h:
typedef enum {
  SrcNone  = 0,
  SrcBP1   = 1,
  SrcSWxy  = 2,
  SrcVRx   = 5,
  SrcVRy   = 6,
  SrcKbd   = 11,
} ySource;
```

yRobot-FreeRTOS

Tasks : **Robot**



MsgX

```
yTask.h:
typedef struct {
  uint8_t Topic;
  float   PayLoadF;
  uint8_t PayLoadI;
} yEvent_t;
```

```
yTask.h:
typedef enum {
  TopicNone  = 0,
  BP1   = 1,
  SWxy  = 2,
  kbd   = 3,
  Mar   = 4,
  Arr   = 5,
  MarArr = 6,
  VRx   = 11,
  VRy   = 12,
  DIR   = 13,
  SP    = 14,
  MotD  = 21,
  MotG  = 22,
  Ro    = 31,
} yTopic;
```

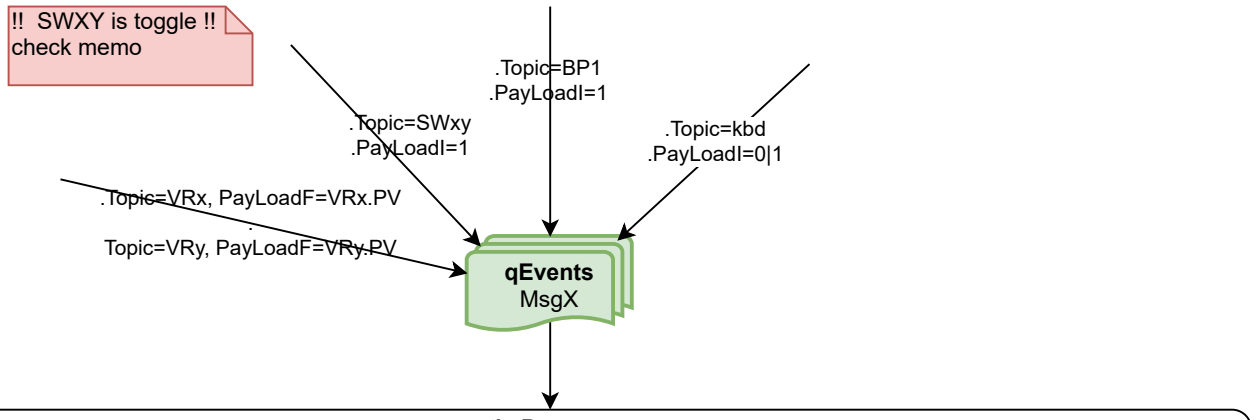
Les tâches:
* attendent un 'event' dans la queue associée.
* font les actions correspondantes
* mettent dans la queue 'qVTaffiche' les résultats des actions

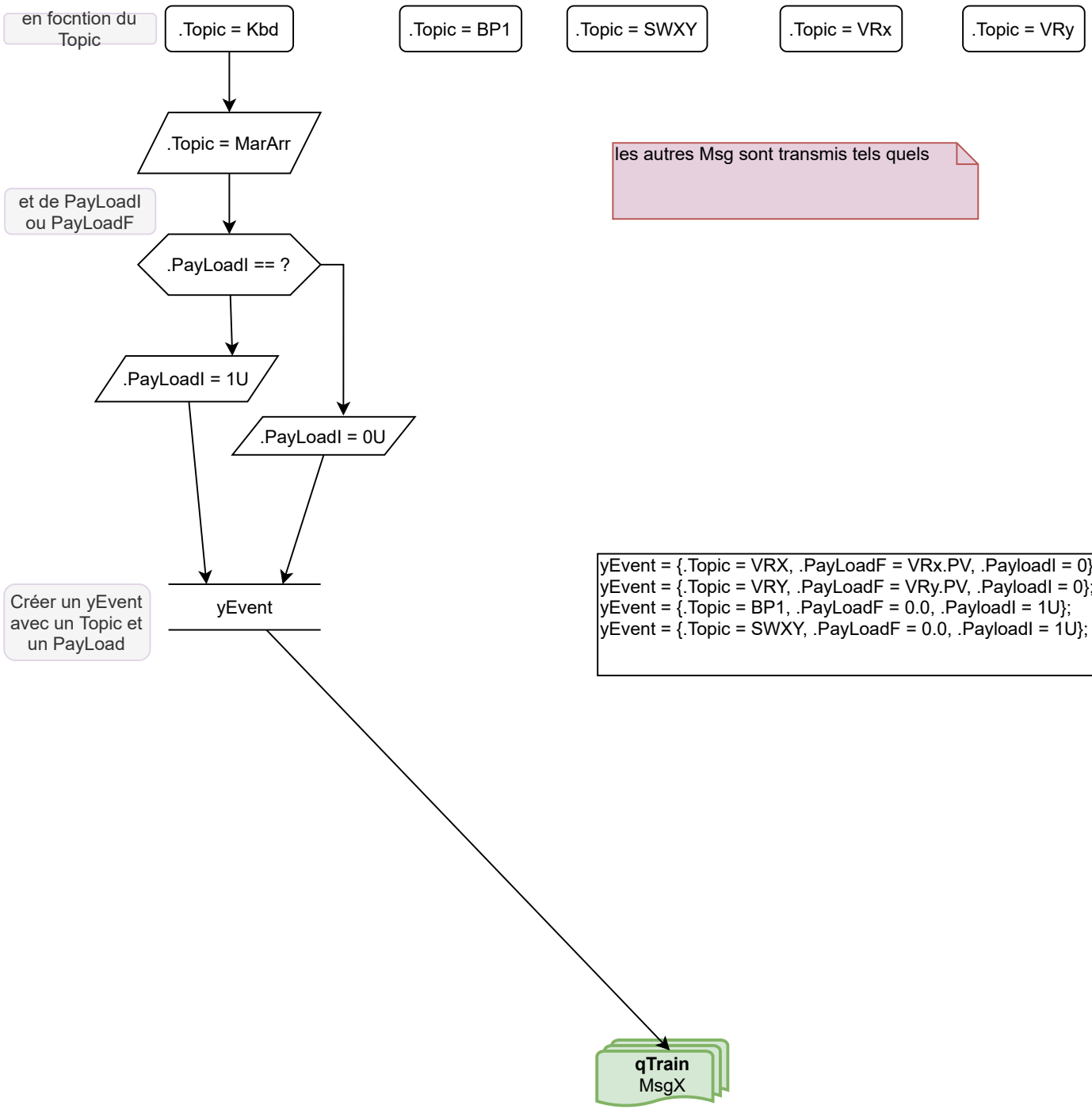
yTRAIN

```
//--- Inputs
    uint8_t MaAr;
    float Vitesse;
    float Direction;
//--- Outputs
    uint8_t inRun;
    uint8_t MaAr_D;
    float Vitesse_D
    uint8_t MaAr_G;
    float Vitesse_G;
//--- Parameters
    float Dir_HH;
    float Dir_LL;
//--- memories
    float Vitesse_memo;
    float Direction_memo;
    uint8_t Sens_memo;
    uint8_t Run_memo;
//--- Virtual outputs
    none
//--- Real outputs
    none
```

yRobot-FreeRTOS

*Task : tk_***Process**





les autres Msg sont transmis tels quels

```
yEvent = {.Topic = VRX, .PayLoadF = VRx.PV, .PayloadI = 0};
yEvent = {.Topic = VRY, .PayLoadF = VRy.PV, .PayloadI = 0};
yEvent = {.Topic = BP1, .PayLoadF = 0.0, .PayloadI = 1U};
yEvent = {.Topic = SWXY, .PayLoadF = 0.0, .PayloadI = 1U};
```

yRobot-FreeRTOS

*Task : tk_***Train**

.Topic=SWxy|VRx|VRY,
.PayLoadF=0|speed,
.PayLoadI=0|1

!! SWXY is toggle !!
check memo

qTrain
MsgX

tk_Train

en focntion du Topic
et de PayLoadI ou PayLoadF

.Topic = SWXY

.Topic = VRx

.Topic = VRy

```
.MaAr = /.MaAr //invert cde

.MaAr_D=.MaAr
.MaAr_G=.MaAr
.inRun=.MaAr
 yEvent{.Topic=MarArr,
          .PayLoadI=.MaAr_D}
   send to qMotD
yEvent{.Topic=MarArr,
          .PayLoadI=.MaAr_G}
   send to qMotG
```

yTRAIN

```
//--- Inputs
    uint8_t MaAr;
    float Vitesse;
    float Direction;
//--- Outputs
    uint8_t inRun;
    uint8_t MaAr_D;
    float Vitesse_D
    uint8_t MaAr_G;
    float Vitesse_G;
//--- Parameters
    float Dir_HH;
    float Dir_LL;
//--- memories
    float Vitesse_memo;
    float Direction_memo;
    uint8_t Sens_memo;
    uint8_t Run_memo;
//--- Virtual outputs
    none
//--- Real outputs
    none
```

yEvent = {.Topic = MarArr, .PayLoadF = 0.0, .PayloadI = EvtRecu.PayloadI};

yEvent

.Topic=Mar|Arr|SP,
.PayLoadF=0|speed,
.PayLoadI=0|1

qMotG
MsgX

qMotD
MsgX

yRobot-FreeRTOS

*Tasks : tk_***Moteur**

.Topic=MarArr|SP,
.PayLoadF=0|speed,
.PayLoadI=0|1

qMotD
MsgX

Idem for
qMotG
MsgX

en focntion du Topic
et de PayLoadI ou PayLoadF

tk_MoteurD

structure 'yMOTOR' 'Moteur_D'

.Topic = MarArr

```
Si .PayLoadI == 1
   .MarArr = 1;
Sinon
   .MArArr = 0;
   .Speed_SP= 0;
   .Sens = 0;
```

yMOTOR

```
//--- Inputs
    uint8_t MarArr;
    float Speed_SP;
//--- Outputs
    uint8_t inRun;
//--- Paramters
    uint32_t DutyCycle;
    float DeadBand;
//--- memories
    float Speed_memo;
    uint8_t Sens_memo;
    uint8_t Run_memo;
    float DB_memo;
    uint8_t FeeWheel;
//--- Virtual outputs
    uint32_t Speed_MV;
    uint8_t  Sens;*
    uint8_t _av;
    uint8_t _ar;
    uint32_t _pwm;
//--- Real outputs
    uint32_t _gpioPortIN1;
    uint16_t _gpioPinIN1;
    uint32_t _gpioPortIN2;
    uint16_t _gpioPinIN2;
    TIM_HandleTypeDef _htimpwm;
```

.Topic = SP

```
Si .MarArr == 1
   .Speed_SP = PayLoadF;
   //check .Speed_SP % .DeadBand
   Si .Speed_SP >= .DeadBand
      .Speed_MV = |.Spedd_SP|;
      .Sens = 1;
      .inRun = 1;
   Si .Speed_SP <= .DeadBand
      .Speed_MV = |.Speed_SP|;
      .Sens = -1;
      .inRun = 1;
   Sinon
      .Speed_MV = 0;
      //.Sens inchangé
      .inRun = 0;
   .Speed_memo = .Speed_SP;
Sinon
   .Speed_MV = 0;
   .Sens = 0;
   .inRun = 0;
//-- re exec motor
```

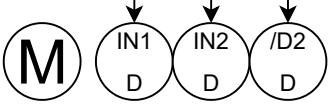
_Exec

```
==> _Virtualoutputs()
Si .Sens a changé
   //- marquer temps d'arrêt
   ._ar, ._av, ._pwm = 0;
   .FreeWheel = 1;
   ==> _RealOutputs()
   osDelay(2ms);
   .FreeWheel = 0;
   ==> _Virtualoutputs()
   ==> _RealOutputs()
Sinon
   .FreeWheel = 0;
   ==> _RealOutputs()
.Sens_memo = .Sens;
```

_VirtualOutputs

```
// en fonction de .Sens
// affecter ._ar, ._av
// et ._pwm
```

_RealOutputs



H bridge	---- MC33886 ----			
	IN1	IN2	D1	/D2
Forward	H	L	0	pwm
Reverse	L	H	0	pwm
FreeWLow	L	L	0	pwm
FreeWHigh	H	H	0	pwm

A voir ??
.Topic = MarArr +
.PayLoadI = 0|1
~~OU~~
~~.Topic = Ma|Ar~~

yRobot-FreeRTOS

tk_Train

après les actions
mettre un msg d'état ds queue

tk_Process

après les actions mettre un msg d'état ds queue

tk_MoteurD

après les actions mettre un msg d'état ds queue

tk_MoteurG

après les actions mettre un msg d'état ds queue

qVTaffiche
MsgY

MsgY

```
yTask.h:
typedef struct
{
  int src;
  char VTbuff[200];
} VTbuffer_t;

VTbuff_t VTbuffer = { .src =
0, .VTbuff = "...\0" };
```

tk_VTaffiche

```
>VTbuff_t BuffAff;
osMessageQueueGet(&BuffAff, maxDelay);
aTXBuffer = BuffAff.VTbuff;
osSemaphoreAcquire (semUART, maxDelay);
HAL_UART_Transmit (aTXbuffer);
osSemaphoreRelease;
```

semUART

defaultTask

```
>BP1Detected = 0;
>secMemo = 0;
>yFlagClearStaus = 0;
osDeay(100ms);
if (yFlagClearStatus++ >= 60) {
    yMenuClearStatusBar();
   yFlagClearStatus = 0;
}

if (BP1Detected != 0) {
    debounce BP1
    invert 'nr_comVTcomGUI'
    VTbuffer.VTbuff = mode VT ou GUI;
    Mettre l'objet 'VTbuffer' dans la queue;
}
Récupérer date & heure;
if (nr_Time.Seconds != secMemo) {
aTXBuffer = formater 'date' & 'heure' pour affichage;
//display direct via sémaphore
 osSemaphoreAcquire (semUART, maxDelay);
 HAL_UART_Transmit (aTXbuffer);
 osSemaphoreRelease;
}
/* pour Node-RED */
nr_w_Compteur += 1;
```

stm32l4xx_it.c

usart2_receive_it
(x caracteres)

uint8_t aRxBuffer[]);

yMenu.c/
yMenuGetTouche

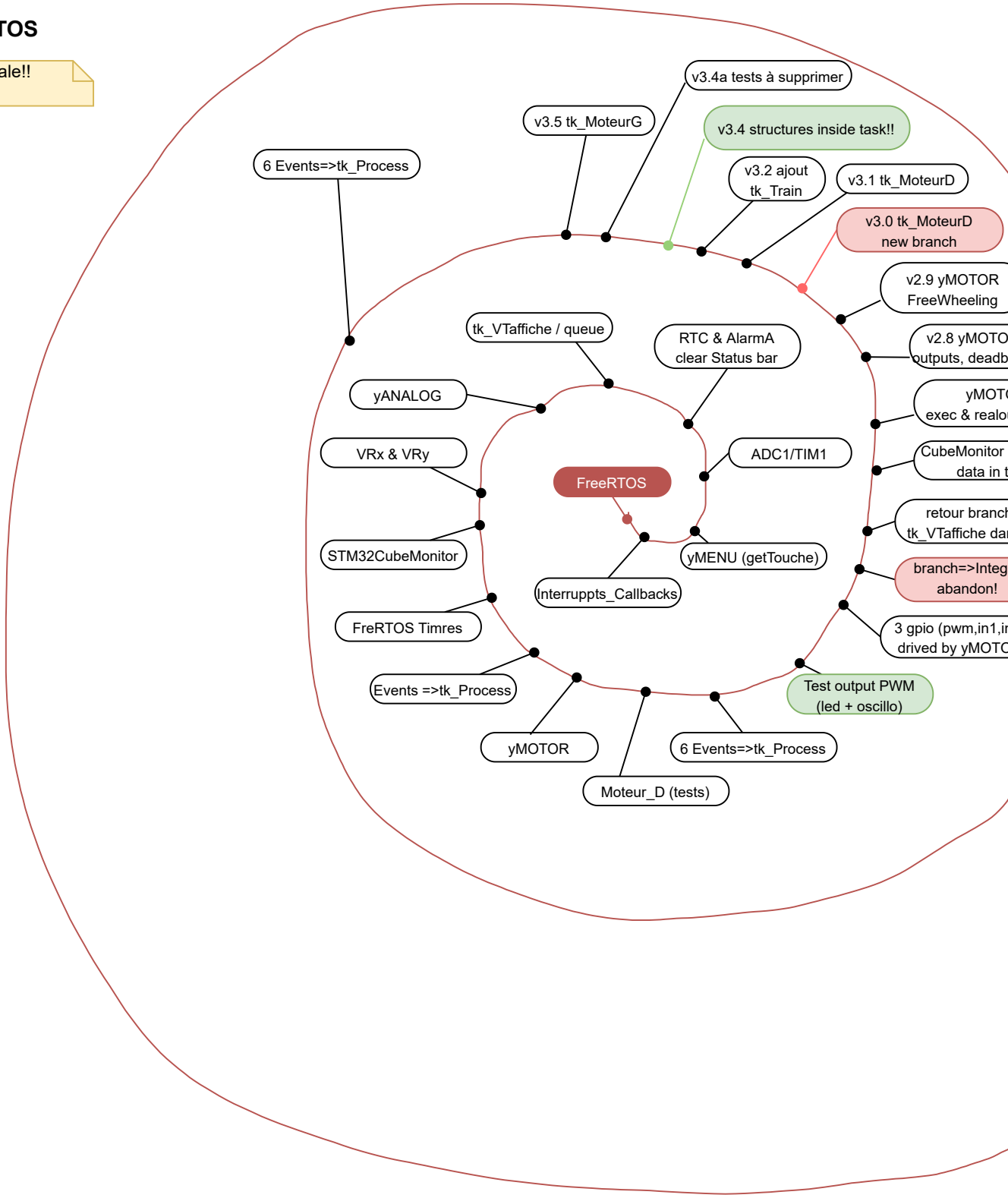
Page-3

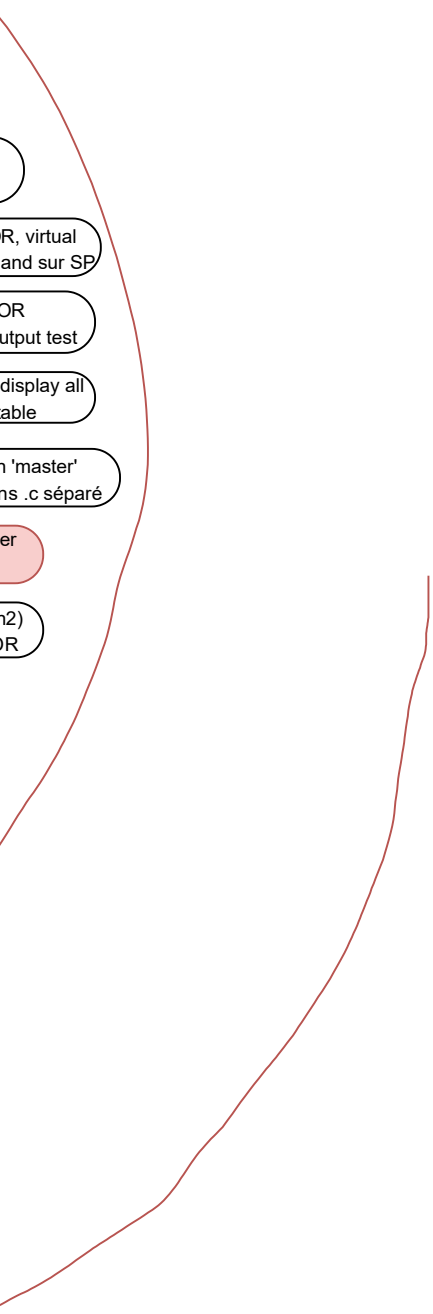
Page-4

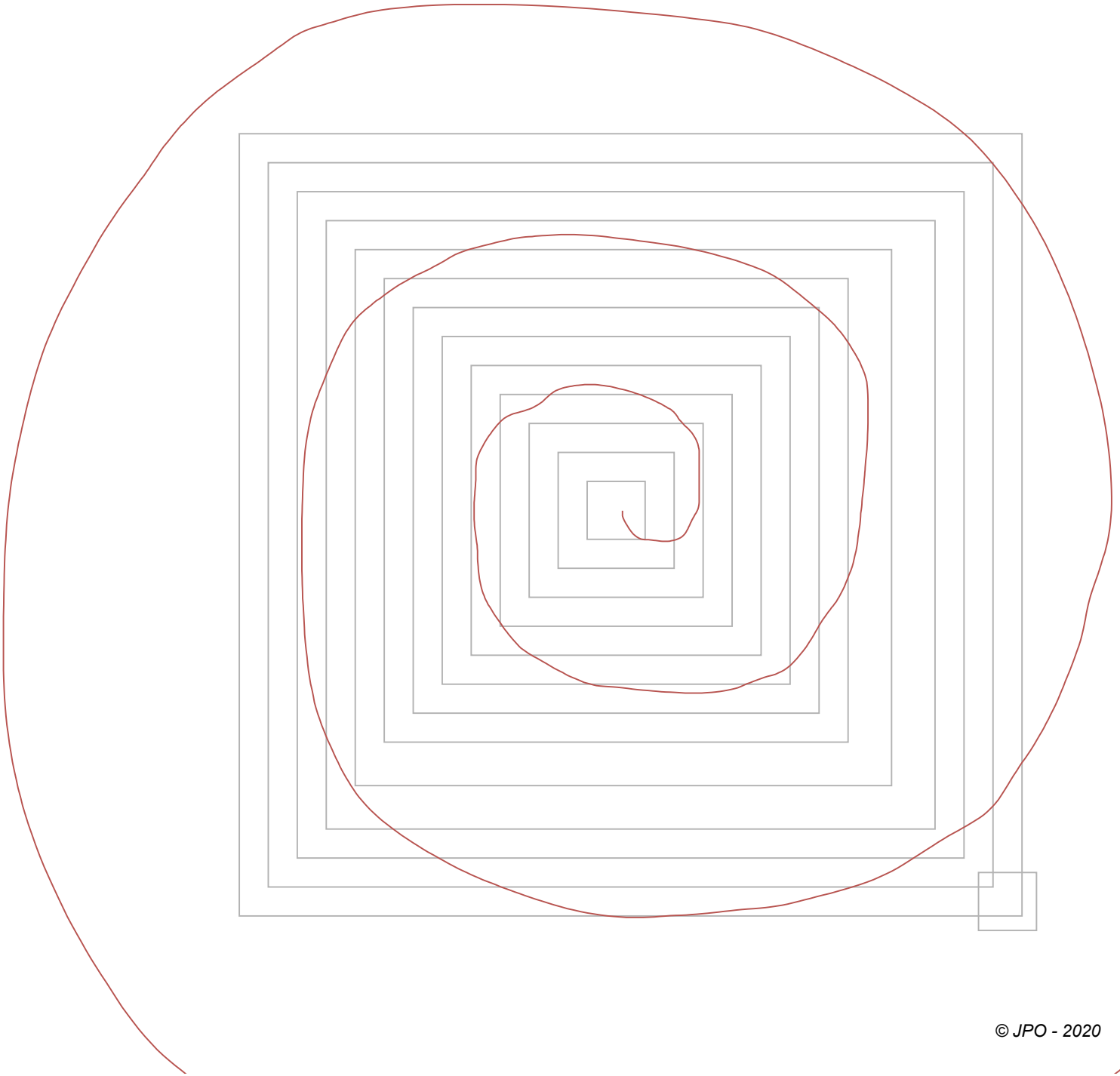
UART génére une interruption.
Le CallbackUart(dsmain.c) appelle 'yMenuGetTouche', quelques actions sont possible mais on est tjrs ds l'interrupt donc il faut aller vite!!

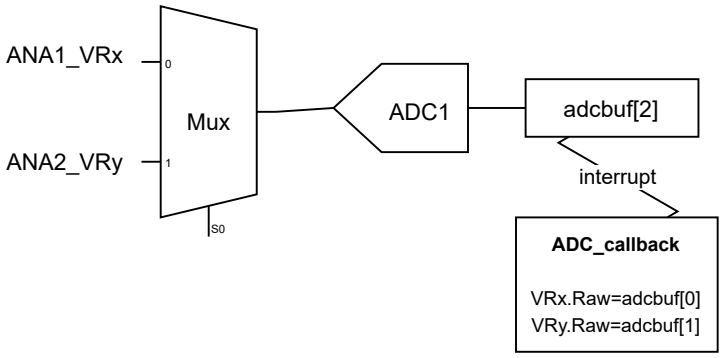
yRobot-FreeRTOS

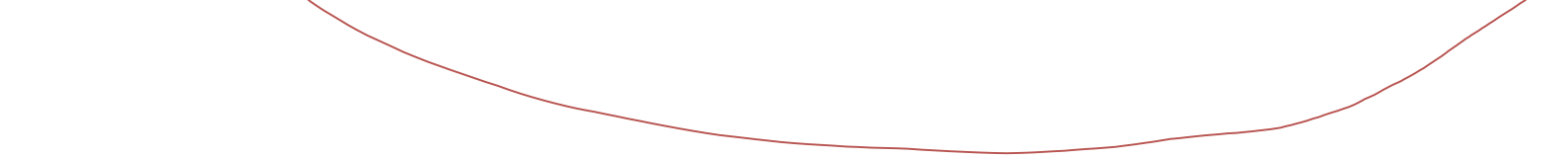
Conception en spirale!!




R, virtual
and sur SP
OR
utput test
display all
able
n 'master'
ns .c séparé
er
n2)
OR

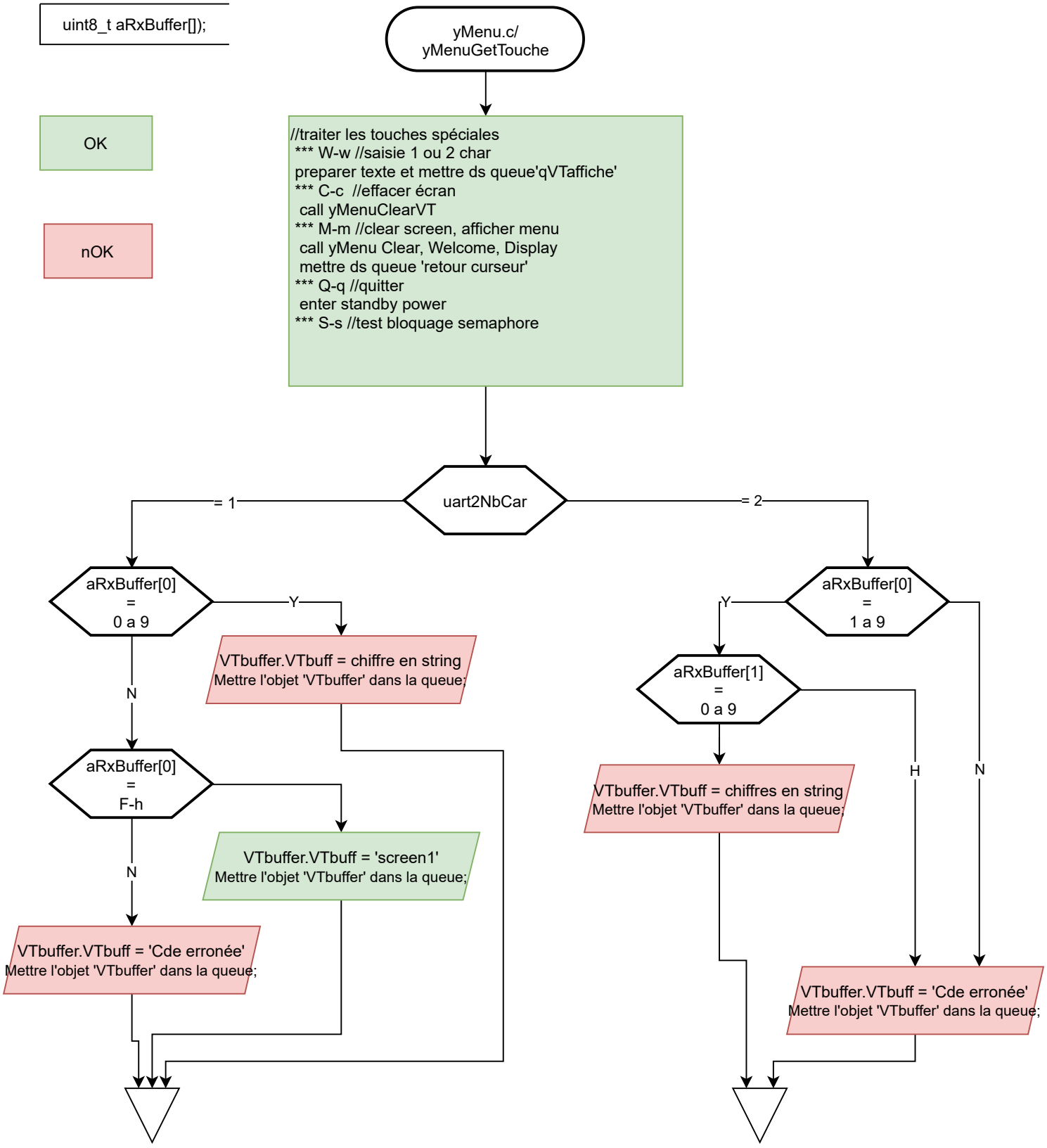



ANA1_VRx
0
Mux
ANA2_VRy
1
S0
ADC1
adcbuf[2]
interrupt
ADC_callback
VRx.Raw=adcbuf[0]
VRy.Raw=adcbuf[1]



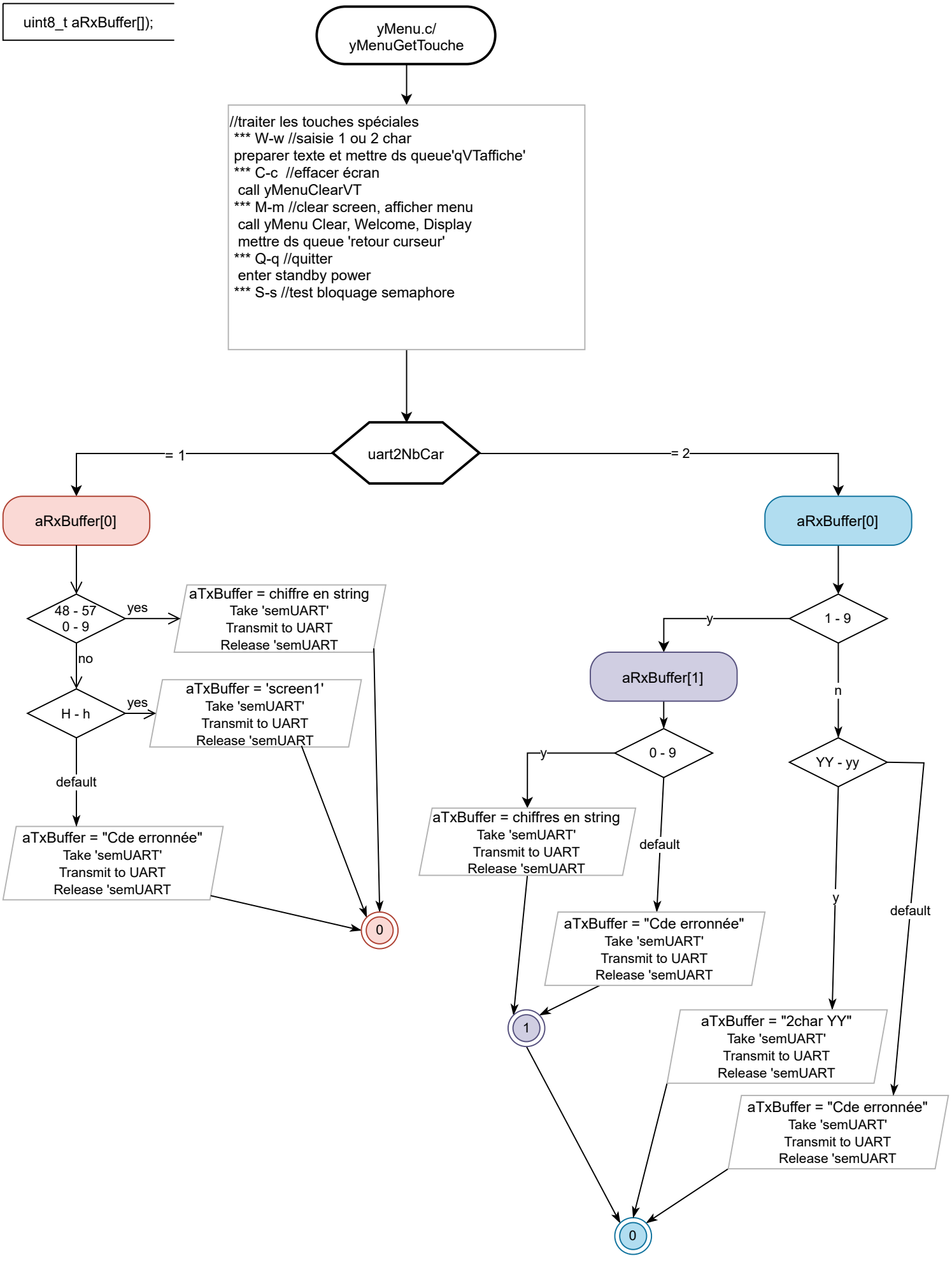
yRobot-FreeRTOS

Détails de la gestion des touches saisies.
S/prog appelé depuis une interruption UART.
Mais qqs pbs!



Détails de la gestion des touches saisies.
S/prog appelé depuis une interruption UART.
Version sans passer par la queue!

uint8_t aRxBuffer[]);

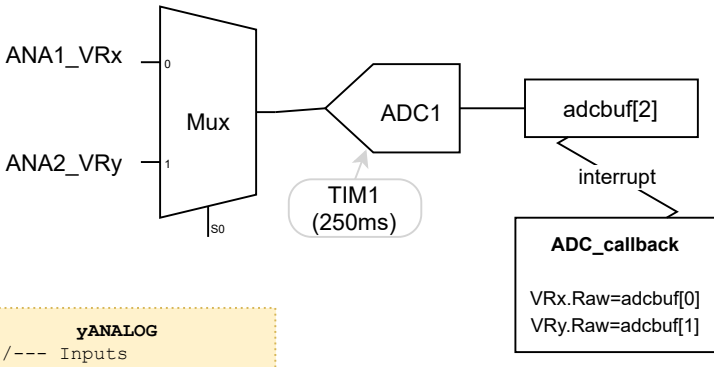


yRobot-FreeRTOS

yANALOG **VRy** ==> *yMOTOR* **Moteur_D**

Note

v2.8 OLD



yANALOG

```
//--- Inputs
    uint32_t Raw;
    uint8_t  Ri;
//--- Outputs
    float    PV;
    float    PVa;
    int8_t   sens;
    uint8_t Ro;
//--- Parameters
    float Ech_Mini;
    float Ech_Maxi;
    float A;
    float B;
    float Coef_Filtre;
    float UnMoinsCoef;
    float    Trim;
    uint32_t TrimRaw;
    float Hysteresis;
//--- Memoires
    float PVmemo;
    float PVhyst;
```

tk_CheckVR

calcul .PV, .PVa, .sens
check variation .Ro
Si .Ro ==> event

.Topic=VRX|VRY
.TopicF=PV
.TopicI=0

qEvents
MsgX

tk_Process

switch Topic
case BP1
case SWxy
case Vrx
case VRy
case kbd

.Topic = SWXY
.PayLoadI = 1U

.Topic=VRY
.PayLoadF=VRy.PV

Moteur_D.inRun
== 1U

Y
yMOTOR_MarArr
(&Moteur_D, yARRET)

N
yMOTOR_MarArr
(&Moteur_D, yMARCHE)

yMOTOR(&Moteur_D, speed)

yMOTOR

```
//--- Inputs
    uint8_t MarArr;*
    float Speed_SP;*
//--- Outputs
    uint8_t inRun;*
    float Velocity;
//--- Paramters
    uint32_t DutyCycle;
    float DeadBand;*
//--- memories
    float Speed_memo;
    uint8_t Sens_memo;
    uint8_t Run_memo;
    float DB_memo;
    uint8_t FeeWheel;
//--- Virtual outputs
    uint32_t Speed_MV;*
    uint8_t  Sens;*
    uint8_t _av;
    uint8_t _ar;
    uint32_t _pwm;
//--- Real outputs
    uint32_t _gpioPortIN1;
    uint16_t _gpioPinIN1;
    uint32_t _gpioPortIN2;
    uint16_t _gpioPinIN2;
    TIM_HandleTypeDef _htimpwm;
```

" Speed modification "
.Speed_SP = speed
si (Speed_SP >= |Deadband|)
.Speed_MV = abs(Speed_PV)
.Sens = signe de Speed_PV
sinon .MV=0; .sens=0;
.Speed_memo=.Speed_SP;

" MarArr request "
.inRun = 0
.Spedd_MV = 0
.Sens=0

" MarArr request "
.inRun = 1

" Exec Motor "
Update VitualsOutputs
Check chgmt de sens
si chgt sens
RAZ virtualoutputs
.FreeWheel=1
Update Real Outputs
delay
.FreeWheel=1
Update VitualsOutputs
Update Real Outputs
sinon
.FreeWheel=0
Update Real Outputs

H bridge ---- MC33886 ----

	IN1	IN2	D1	/D2
Forward	H	L	0	pwm
Reverse	L	H	0	pwm
FreeWLow	L	L	0	pwm
FreeWHigh	H	H	0	pwm

IN1 = H ==> OUT1 = H
IN2 = H ==> OUT2 = H
FreeWhellHigh = frein

" Virtual Outputs "
Update _av, _ar, Speed_MV, Sens
seulement si iRun=1
si Sens=1 => _av=1;_ar=0;
si Sens=-1 ==> _av=0; _ar=1;
si Sens=0 ==> _av=0; _ar=0;
Sens_memo=Sens;
_pwm=Speed_MV;

sinon
_ar=0; _av=0, _pwm=0;

" Real Outputs "
Update _IN1, _IN2, _htimpwm
_IN1=_av &!_ar
_IN2=_ar & !_av
_htimpwm=_pwm

tk_Default

//--- traces
LD2 = Motor_D.inRun
Display états Moteur_D via tk_VT