Mind your distance! Adding wireless control to TI-Rover

1. Project description

Develop a concept to implement a wireless R/C model control system to control speed and direction of TI-Rover, meeting requirements as follows:

a. Use a OEM Spektrum DSMx remote control using PWM to transmit control stick position to Rover
b. Convert DSMx PWM signals to analog voltage levels
c. Use IN1/2/3 as analog voltage inputs for TI-Rover
d. Create a Control Stick calibration program which allows universal adaptation of individual analog input voltage level swing and offset
e. Create a main control program using the parameters as generated by the Control Stick calibration program, supporting features as follows:
    i. TI-Rover should not move when Gas and Steering stick is in neutral position
    ii. TI-Rover speed and direction should be proportional to Gas stick position and direction (forward/backward)
    iii. TI-Rover turn radius should be proportional to Steering stick position and direction (left/right)

2. Spektrum DSMx components

A DX6i 2.4GHz transmitter and a 4-channel AR400 Sport Receiver are used as shown here. Any model R/C transceiver system creating a Pulse Width Modulation (PWM) output at the receiver side with a minimum of 2 channels can be used.

Figure 1 – Spektrum Transmitter

Figure 2 – Spektrum Receiver

The PWM Protocol creates a square wave signal per channel, at a frequency of about 50Hz. Stick position is translated into the pulse width of this square wave, ranging from 1000us up to 2000us. Center point (sticks are in neutral position) is at about 1500us.

3. The Receiver-to-Rover Hardware interface challenge

All BB pins of the TI-Innovator Hub residing in the TI-Rover are used up by the Hub-to-Rover interface. 3 input ports (IN1/2/3) and 3 output ports (OUT1/2/3) are the only interface ports available in the TI-Rover application.
Currently, there is no reliable and fast method implemented in the TI-Hub firmware to measure digital input pulse width generated by a PWM signal. An adapter is needed to convert the PWM signal of the DSMx receiver to an electrical signal the TI-Hub can measure precisely and fast. This led to the PWM-to-Analog converter chip LTC2644 designed by Analog Devices.
The DSMx receiver's PWM output signal frequency of only 50Hz, paired with a PWM range of 1000us-2000us results in a rather low PWM swing of 4.5-9%. This results in only 9% of the total available output voltage swing of the LTC2644 can be used in this application. Therefore, the 3.3V IN1 and IN2 ports are used, to maximize utilization of the resolution of the TI-Hub built-in Analog-to-Digital converter.

Last but not least, the Spektrum AR400 receiver's minimum Vcc is 5V, so a third interface cable to the Hub's IN3 port is needed, to be the power source for both, AR400 and the LTC2644.

4. Wireless Interface Board

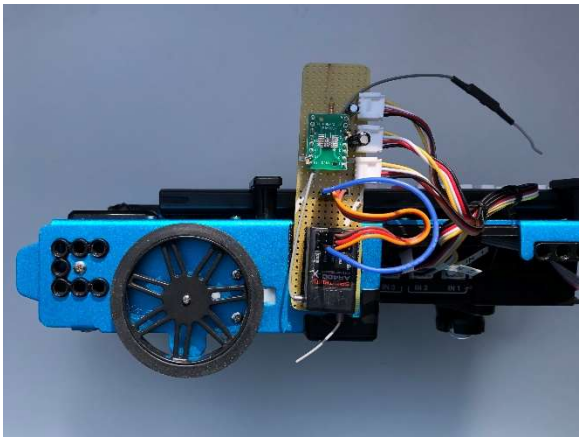

Figure 3 – Velcro mount of I/F board

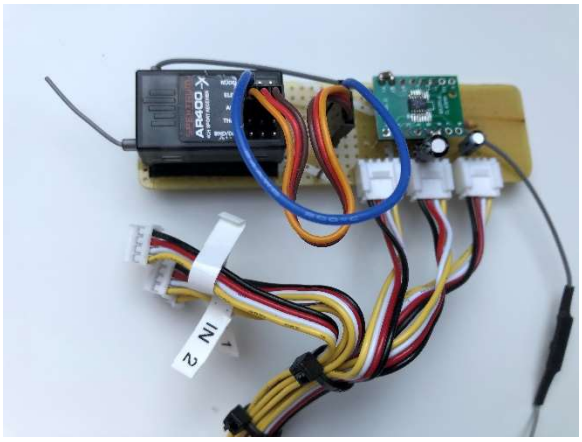

Figure 4 – I/F board

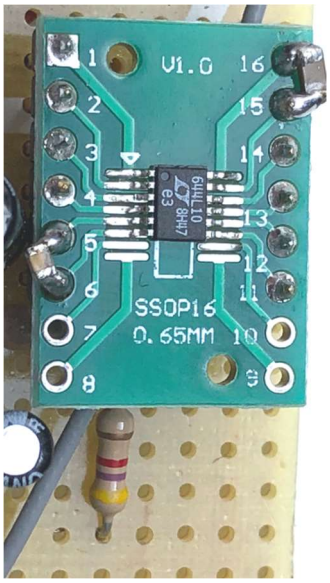

Figure 5 – LTC2644 mounted on breakout board along with SMD Capacitors

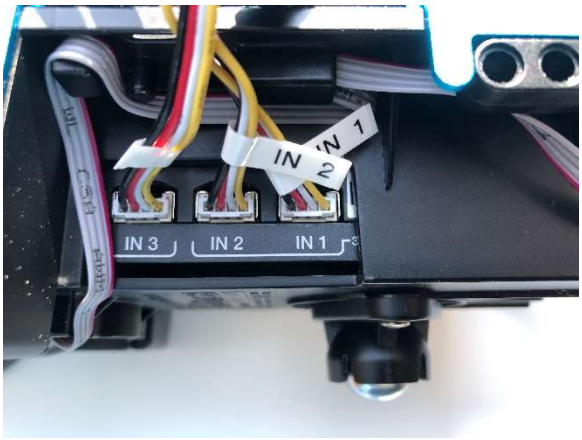

Figure 6 – Wiring I/F board to TI-Hub

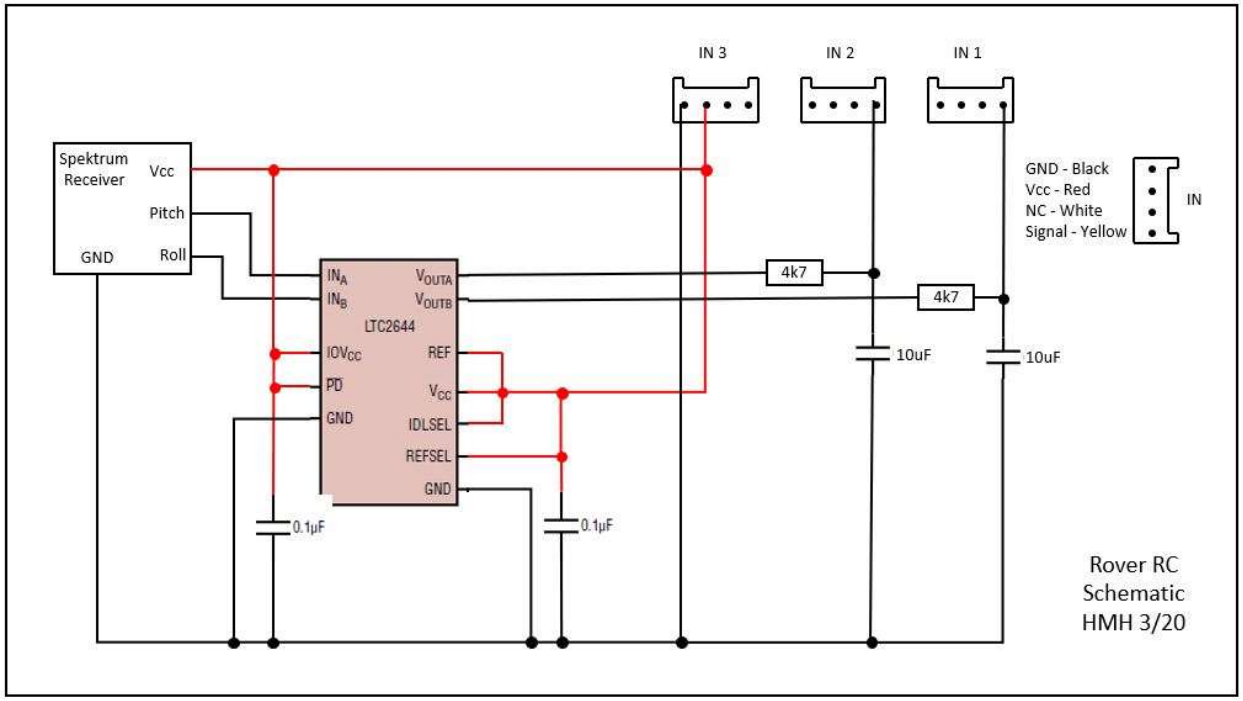

Figure 7 – Interface Board Schematic

The interface board is built on a prototype board, with all leads manually soldered (no PCB traces). Figure 3 to 6 illustrate the wiring and Velcro mount to the TI-Rover's side.
Unfortunately, there is no Grove board available supporting the LTC2644, so a breakout board is used to adapt the MSOP surface mount package of the LTC2644 to a dual-in-line throughhole-mount package form factor.
The 4k7 / 10uF components connected to VoutA and VoutB outputs are used as a low pass filter to reduce the PWM jitter of the AR400, which would otherwise be translated into an analog signal jitter fed into the TI-Hub.

5. TI-Innovator Hub Programs

The TI-Innovator Hub Program consists of three important elements:

- calsticks() Program to calibrate analog signal swing, center points, noise/deadband
- main() Program to control TI-Rover motors via the Send "Set rv.motors xxx yyy" command
- map(x,in_min,in_max,out_min,out_max) function to translate analog voltages to motor PWM values

See code listings in the Appendix.

6. calsticks() Program

Analog voltage for controlling Speed and Direction (forward/backward) of the TI-Rover is supplied into IN1. Maximum analog voltage will result in maximum speed forward, minimum analog voltage will result in maximum speed backward. Stick neutral will be in between max and min input voltage and result in TI-Rover not moving.
Analog voltage for controlling turns (steering) of the TI-Rover is supplied into IN2.
There are 5 turn behavioral states:

a. Steering stick is at neutral: both wheels spin at the same speed (either forward or backward, depending on gas stick)
b. Steering stick is moved to less than 50% to the left: right wheel will spin at gas stick speed, but left wheel will turn at a slower speed, down to zero speed at 50% stick position left .
c. Steering stick is moved to more than 50% to the left: right wheel will continue spinning at Gas stick speed, left wheel will start spinning in the opposite direction as before, ending up at same speed of spin, opposite direction, when steering stick is fully moved to the left. Rover turns around its front wheel axis' center point.
d. Steering stick is moved to less than 50% to the right: same as under (b) above, but opposite wheels
e. Steering stick is moved to more than 50% to the right: same as under (c) above, but opposite wheels

Gas stick will control the speed of the turn overall.
calsticks() will measure center points, minimum and maximum stick analog voltages and necessary deadband needed to filter system noise when sticks are in neutral and extreme positions and store these values in global variables to be used by the main() program.

calsticks() is cycling through 3 steps:

a. Measure center points and noise of both, gas and steering input voltages. Do not touch sticks during that phase. Observe the 'jitter' for gas and steering on the display. If you do not see a change in minimum and maximum values, proceed to the next step by pressing any key on the nSpire handheld.
b. Move Gas stick to the minimum and maximum position. It does not matter whichever you do first. Observe min and max gas values. If you no longer see a change, proceed to the next step by pressing any key on the handheld.
c. Move Steering stick to the minimum and maximum position. Press any key to finish the calsticks() program.

A maximum deadband of +/- 8 digits is still acceptable, a good analog voltage supply should have a deadband of not more than +/-3 digits. If you encounter higher values, you should tune your analog system by changing the low pass filter components of the interface board.

7. main() program

main() uses the global variables created by calsticks(). It's essential that execution of main() is preceeded by a proper calsticks() run. Here's a list of the global variables and their meaning:

- dbandgl: Gas neutral lower Deadband
- dbandgh: Gas neutral upper Deadband
- dbandsl: Steering neutral lower Deadband
- dbandsh: Steering neutral upper Deadband
- ming: Rover full speed backward
- maxg: Rover full speed forward
- mins: minimum radius left turn
- maxs: minimum radius right turn

There is a <debug> switch variable (line 8) which allows the print out of a couple key variables at run time, if a system debug is needed. <debug> mode will slow down the entire Rover update loop, resulting in a longer latency and less responsive behavior of the TI-Rover to stick movements. In normal operation, <debug> should be set to <false>.

main() supports stick reversal, in order to support other analog voltage systems/sensors than the Spektrum system described here. For example, you could use an ADXL335 Grove sensor to supply analog voltage based on sensor tilt information into IN1/2. This may require a 'stick reversal' of either Gas or Steering or both, depending on the way the Grove sensor is held or mounted. The same may apply when using the 3-axis accelerometer from Vernier.

If <reverseg> and <reverses> are set to <false>, the following input voltage to control behavior applies:

- low gas: Rover full speed backwards
- high gas: Rover full speed forward
- low steering: Rover full left turn
- high steering: Rover full right turn

Setting <reverseg> to <true> would result in low gas – full speed forward and high gas – full speed backward.

8. map() function

The map() function is a powerful math function to scale virtually any input range to any output range, as long as the input-to-output behavior has a linear dependency. This makes the Rover main() program so versatile. The program does not need to know the absolute analog gas input voltage or which sensor feeds the analog voltage. It just translates whatever the analog input voltage range is into the 8-bit (0-255) range of the Rovers rv.motors motor control command. Even better, for steering control it 'automatically' translates the complex inner wheel movement behavior based on the steering stick, from 255-to-0-to-negative255.

9. Appendix – code listing

```
Define calsticks()=                                   Stick calibration program
Prgm
:
:Local i,firstrun,gas,steer,dbandg,dbands,stat
:
:Send "CONNECT ANALOG.IN 1 IN 1"                      Gas input
:Send "CONNECT ANALOG.IN 2 IN 2"                      Steering input
:
:firstrun:=true
:
:DispAt 1,"Measuring Deadband, don't move sticks"
:While getKey()=""                                    Step 1: Measure center points and noise
: If firstrun=true Then                               Get initial values to set initial min and max values
:   Send "READ ANALOG.IN 1"
:   Get gas,stat
:   While stat=0                                      Sometimes Hub is slow on the initial read, this is to
:     Get gas,stat                                    avoid a 'variable error' at runtime
:   EndWhile
:   Send "READ ANALOG.IN 2"
:   Get steer
:   ming:=gas
:   maxg:=gas
:   mins:=steer
:   maxs:=steer
:   firstrun:=false
: EndIf
: Send "READ ANALOG.IN 1"                             Here the center point/deadband loop starts
: Get gas
: Send "READ ANALOG.IN 2"
: Get steer
: If gas>maxg Then
:   maxg:=gas
: ElseIf gas<ming Then
:   ming:=gas
: EndIf
: If steer>maxs Then
:   maxs:=steer
: ElseIf steer<mins Then
:   mins:=steer
: EndIf
: DispAt 2,"Gas: ",gas,"Min:",ming," Max: ",maxg
: DispAt 3,"Steer: ",steer,"Min:",mins," Max: ",maxs
:EndWhile                                             Key is pressed, calculate deadband and global variables
:dbandg:=int(((maxg-ming)/(2)))
:dbands:=int(((maxs-mins)/(2)))
:dbandgl:=ming
:dbandgh:=maxg
:dbandsl:=mins
:dbandsh:=maxs
:
:DispAt 1,"Gas ","Deadband: +/- ",dbandg              Display results
:DispAt 2,"Steering ","Deadband: +/- ",dbands
:
```

```
:DispAt 3,"Move Gas forward, then backward."
:While getKey()=""
:  Send "READ ANALOG.IN 1"
:  Get gas
:  If gas>maxg Then
:    maxg:=gas
:  ElseIf gas<ming Then
:    ming:=gas
:  EndIf
:  DispAt 4,"Gas: ",gas,"Min:",ming," Max: ",maxg
:EndWhile
:ming:=ming+2*dbandg
:maxg:=maxg-2*dbandg
:DispAt 3,"Gas: min:",ming," max: ",maxg
:DispAt 4,"Move Steer to Left, then Right."
:While getKey()=""
:  Send "READ ANALOG.IN 2"
:  Get steer
:  If steer>maxs Then
:    maxs:=steer
:  ElseIf steer<mins Then
:    mins:=steer
:  EndIf
:  DispAt 5,"Steer: ",steer," min: ",mins," max: ",maxs
:EndWhile
:mins:=mins+2*dbands
:maxs:=maxs-2*dbands
:DispAt 4,"Steer: min:",mins," max: ",maxs
:DispAt 5,"done!"
:EndPrgm
```

Step 2: calibrate min & max gas stick positions

Guardband to ensure max forward & backward speeds in spite of PWM noise

Step 3: calibrate min & max steering stick positions

Guardband to ensure tightest turns in spite of PWM noise

```
Define main()=                                          main Rover control program
Prgm
:Local dbug,gas,steer,backward,reverseg,reverses,mot,motl,motr
:
:Send "CONNECT ANALOG.IN 1 IN 1"                        Analog gas input
:Send "CONNECT ANALOG.IN 2 IN 2"                        Analog steering input
:
:Send "CONNECT RV"
:
:dbug:=false                                            Set dbug to 'true' to display key variables at run time, pls note this
:                                                       will increase latency of the control loop!
:reverseg:=false                                        Set to 'true' if gas stick reversal is needed
:reverses:=false                                        Set to 'true' if steering stick reversal is needed
:
:backward:=false                                        Flag to indicate Rover being in backwards motion
:
:Disp "Running...! (dbug= )",dbug
:
:While getKey()=""                                      Main loop
: mot:=0                                                If Gas stick is in neutral, don't move motors
: motl:=0
: motr:=0
: Send "READ ANALOG.IN 1"
: Get gas
: Send "READ ANALOG.IN 2"
: Get steer
: If gas≥dbandgh Then                                   Gas is exceeding center value + guardband
:   If reverseg=false Then                              Check for stick reversal flag
:     backward:=false
:   Else
:     backward:=true
:   EndIf
:   mot:=map(gas,dbandgh,maxg,0,255)                    Map gas stick value to motor PWM, forward motion
: ElseIf gas≤dbandgl Then                               The same procedure for backward motion
:   If reverseg=false Then
:     backward:=true
:   Else
:     backward:=false
:   EndIf
:   mot:=map(gas,ming,dbandgl,255,0)                    Map gas stick value to motor PWM, backward motion
: Else
:   mot:=0                                              Don't move motors if within center value +/- guardband
: EndIf
: If steer≥dbandsh Then                                 Voltage exceeds center value + guardband
:   motl:=mot                                           Start a right turn, left wheel spins at 'gas' speed
:   motr:=map(steer,dbandsh,maxs,mot,−mot)              Right wheel slows down or spins reverse, depending on stick
: ElseIf steer≤dbandsl Then                             Same procedure on a left turn
:   motr:=mot                                           Right wheel continues to spin at gas speed
:   motl:=map(steer,mins,dbandsl,−mot,mot)              Left wheel slows down or spins reverse
: Else
:   motr:=mot                                           Right and left wheel spin at same speed, if steering stick is at
:   motl:=mot                                           neutral
: EndIf
: If reverses=true Then                                 Steering stick reversal?
:   mot:=motl                                           Yes: swap between left and right motors
:   motl:=motr
:   motr:=mot
: EndIf
```

```
:  If dbug=true Then                                          If debug flag is set,
:    DispAt 1,"Gas ",mot                                      Display gas stick motor value
:    DispAt 2,"backward: ",backward                           Display forward/reverse flag value
:    DispAt 3,"gas reverse: ",reverseg,", steer reverse: ",reverses   Display stick reversal values
:    DispAt 4,"motl: ",motl," motr: ",motr                    Display left and right motor values
:  EndIf
:  If backward=false Then
:   Send "SET RV.MOTORS eval(-motl) eval(motr)"               Update spin speed of left and right motors
:  Else
:    Send "SET RV.MOTORS eval(motl) eval(-motr)"
:  EndIf
:EndWhile
:Send "SET RV.MOTORS 0 0"                                     If any key is pressed on the HH, stop motors
:Send "DISCONNECT RV"                                         And disconnect Hub from RV
:EndPrgm



Define map(x,in_min,in_max,out_min,out_max)=                  Range scaling/mapping function
Func
:
:If x<in_min Then                                             Clamp input to in_min & in_max values to avoid mapping outside
:  x:=in_min                                                  given input range
:EndIf
:If x>in_max Then
:  x:=in_max
:EndIf
:
:Return int((((x-in_min)*(out_max-out_min))/(in_max-          Return mapped value
in_min))+out_min)
:EndFunc
```