dsNav communication

In order to allow a quick developing of a communication program with dsNav in whatever language, a vey simple communication protocol has been implemented.
The dsNav board is a “slave” from the communication point of view. It is waiting for commands from COMM1 or COMM2 and never starts a communication sequence by itself.
After the whole command packet has been received and verified, an internal action is performed by dsNav or the requested parameters are sent back to the master as required by that specific command.
If the command is unknown or if the packet is not coherent (invalid length or wrong checksum) it is discarded and no action is performed.
Some examples will be provided in some free development environments such Arduino (http://www.arduino.cc) for a microcontroller board, and Processing (http://www.processing.org) for any kind of computer operating system. They are so simple and wide open that can be considered as a meta-language easy to be converted in other languages.
A full working Processing computer program that acts as a complete console will be provided as well to immediately start configuring and working with dsNav board.

Commands.
Command strings are variable length type. Any string is composed by an array of N unsigned char:

Pos.	*Name*	*Values range*	*Description*
0	Header	@	Always present. Needed to synchronize communication
1	Id	0-9 ASCII	The id of the board addressed by this command. Not used here, may be used if more devices are present
2	Cmd	ASCII	All ASCII printable characters may be used
3	CmdLen	Binary	Number (N) of bytes following (checksum included)
		...	data 0
		...	
		...	data x
N	Checksum	0-255	obtained by simply adding up in a 8 bit variable, all bytes that compose the message (checksum itself excluded)

*Arduino example for command **d***

- w

This is a write only command, it sends to dsNav the values read from the sensor board regarding the distance from the obstacles on Left, Center and Right side; the distance from the targets (light sources) on Left, Center and Right side and the value read from digital compass as an integer (2 bytes).

- TX:

CmdLen = 9 , Params 8 (6 bytes + 1 int)

(6 char values one byte each + 1 int value + checksum)

Example in Arduino language:

```
void DataPrint(void)
{// send acquired data on serial port to Navigation Control Board
  int i;                        // generic index
  byte TxBuff[16];
  byte ChkSum=0;
  TxBuff[0] = 64;               // Header = "@"
  TxBuff[1] = 0;                // Id = 0
  TxBuff[2] = 100;              // command = "d"
  TxBuff[3] = 9;                // command length
  TxBuff[4] = Dist[L];          // data 0
  TxBuff[5] = Dist[C];          // data 1
  TxBuff[6] = Dist[R];          // data 2
  TxBuff[7] = Target(L);        // data 3
  TxBuff[8] = Target(C);        // data 4
  TxBuff[9] = Target(R);        // data 5
  TxBuff[10]= (byte)(CmpBearing >> 8);// data 6
  TxBuff[11]= (byte)(CmpBearing); // data 7 (data 6 + 7 = int)
  for (i=0; i<=11; i++)
  {
    ChkSum += TxBuff[i];
    Serial.print(TxBuff[i], BYTE);
  }
  Serial.print(ChkSum);
}
```

- RX:

N/A

(The sender doesn't wait for any parameter back)

TX
An example on how to realize a generic command sending procedure with Processing language.

```
byte[] TxBuff = new byte[144];  // Transmission buffer
int TxHeadLen = 4;              // TX header length (@ + Id + Cmd + CmdLen)
int[] TxIntValue = new int[128]; // int value buffer

void TxData(int Id, int Cmd, int ValueLen, int IntFlag)
{/* Transmit a string toward dsNavCon board
       Id       = address of the device, always 0 in our configuration
       Cmd      = Command as described later
       ValueLen = the number of parameters to send
       IntFlag  = the type of the parameters to send
*/
  int TxBuffCount, ChkSum = 0, CmdLen = 0;

  if (IntFlag == 0)  // IntFlag=0 means that a byte variable has to be sent
  {
    CmdLen = ValueLen;
    for (TxBuffCount = 0; TxBuffCount < ValueLen; TxBuffCount++)
    {
      TxBuff[(TxBuffCount*2)+TxHeadLen] = (byte)(TxIntValue[TxBuffCount]);
    }
  }
  if (IntFlag == 1)  // an integer variable has to be sent
  {
    CmdLen = ValueLen*2;          // 1 int value = 2 bytes to transmit
    for (TxBuffCount = 0; TxBuffCount < ValueLen; TxBuffCount++)
    {
      TxBuff[(TxBuffCount*2)+TxHeadLen] = (byte)(TxIntValue[TxBuffCount] >> 8);
      TxBuff[(TxBuffCount*2)+TxHeadLen+1] = (byte)(TxIntValue[TxBuffCount]);
    }
  }
  if (IntFlag == 2) // a long variable has to be sent
  {
    CmdLen = ValueLen*4;        // 1 long value = 4 bytes to transmit
    for (TxBuffCount = 0; TxBuffCount < ValueLen; TxBuffCount++)
    {
      TxBuff[(TxBuffCount*4)+TxHeadLen] = (byte)(TxIntValue[TxBuffCount] >> 24);
      TxBuff[(TxBuffCount*4)+TxHeadLen+1] = (byte)(TxIntValue[TxBuffCount] >> 16);
      TxBuff[(TxBuffCount*4)+TxHeadLen+2] = (byte)(TxIntValue[TxBuffCount] >> 8);
      TxBuff[(TxBuffCount*4)+TxHeadLen+3] = (byte)(TxIntValue[TxBuffCount]);
    }
  }
  TxBuff[0] = (byte)('@');
  TxBuff[1] = (byte)(Id);
  TxBuff[2] = (byte)(Cmd);
  TxBuff[3] = (byte)(CmdLen+1);  // included CheckSum
  for (i=0;i<(TxHeadLen+CmdLen);i++)
  {
   ChkSum += TxBuff[i];
  }
  TxBuff[TxHeadLen+CmdLen] = (byte)(ChkSum);

  for (i=0;i<(TxHeadLen+CmdLen+1);i++)
  {
    RS232Port.write(TxBuff[i]);
  }
 }
```

After filling `TxIntValue[]` buffer with the desired values, the above described `TxData()` procedure can be called.
Example:

```
TxIntValue[0] = DesCoordX;
TxIntValue[1] = DesCoordY;
TxData(0, 'P', 2, 1);
```

Means: send to device with **Id=0** the command **'P'** gettting **2** values from TX buffer as **integer**

RX

An example of a function usable to receive commands from dsNav. This function controls the command packet received for coherence (correct header + known command + checksum ok) and returns *TRUE* to the calling procedure only if everything's ok:

```
boolean RxData(char Cmd,int Len)
{
  int ChkSum = 0;
  for (i=0; i < Len+HeadLen+1; i++) // loops for all data expected and only for that
  {
     RxBuff[i] = (RS232Port.read()); // fills the receive buffer
  }

  if (RxBuff[0] != '@') // is the header correct, is that the correct byte
sequence ?
  {
     RS232Port.clear();
     return false;
  }
  else if (RxBuff[2] != Cmd) // is that the expected command ?
  {
     RS232Port.clear();
     return false;
  }
  else if (RxBuff[3] != (Len+1)) // does it send the right number of characters ?
  {
     RS232Port.clear();
     return false;
  }

  for (i=0; i < Len+HeadLen; i++)  //  ChkSum excluded
  {
    ChkSum += (char)(RxBuff[i]);
  }
  ChkSum = ChkSum % 256;
  if (RxBuff[i] != ChkSum) // is the received checksum equal to the computed one ?
  {
    RS232Port.clear();
   return false;
  }
/* if everything's ok the buffer is analyzed for the expected data by the calling
procedure, otherwise the serial port is cleared and a FALSE condition is returned
*/
  return true;
}
```

Calling the above `RxData()` procedure the RX buffer is filled with the received data. If the function returns TRUE to the calling procedure, this buffer is analyzed for the expected values.

Example:

```
if (RxData('A',13))
{
 MesSpeed = ((RxBuff[HeadLen] << 8(RxBuff[HeadLen+1]));// two bytes form an integer
variable
 Current = (float)((RxBuff[HeadLen+2] << 8) + (RxBuff[HeadLen+3]));
 Xpos = ((RxBuff[HeadLen+4] << 8) + (RxBuff[HeadLen+5]));
 Ypos = ((RxBuff[HeadLen+6] << 8) + (RxBuff[HeadLen+7]));
 MesAngle = ((RxBuff[HeadLen+8] << 8) + (RxBuff[HeadLen+9]));
 CompassAngle = ((RxBuff[HeadLen+10] << 8) + (RxBuff[HeadLen+11])) / 10;
 IdlePerc = RxBuff[HeadLen+10];
}
```

Commands grouped by function type:

Commands (Cmd field) are one single character long and case sensitive.
In following list:

- **r** this is a command that request parameters reading from dsNav to the master
- **w** this is a command that writes parameters to dsNav from the master
- **TX** these are the parameters sent from supervisor and received by dsNav
- **RX** these are the data sent back by dsNav after a correct decoding of the command and received by external board or PC.

The TxData and RxData procedures will be used as example. The code examples are written in Processing language on the PC side.

--- *Service commands*

#
- w
Start scheduler sequence
Read detailed explanations in manual

- TX:
  CmdLen = 2 Params 1
  Example:
  ```
  TxData(0, '#', 0, 1);
  ```
- RX:
  N/A

*
- w
Board software reset
For security reason, this command has to be sent three times consecutively in order to perform reset

- TX:
  CmdLen = 1 Params 0
  Example:
  ```
  TxData(0, '*', 0, 1);
  ```
- RX:
  N/A

$
- r
One row of map grid sending request

- TX:
  CmdLen = 2 Params 1
  Example:
  ```
  TxData(0, '$', 1, 1);
  ```
- RX:
  CmdLen = 80 Params 81
  Example:

```
 if (RxData('$',MapXsizeR+6))
{
MapSendIndx = RxBuff[HeadLen]; // normalized index of the current row on field map matrix
Xshift = Int16toint32(((RxBuff[HeadLen+1] << 8) + (RxBuff[HeadLen+2])));
Yshift = Int16toint32(((RxBuff[HeadLen+3] << 8) + (RxBuff[HeadLen+4])));
Ycoord = RealCoord(Yindx, Yshift, MinMapY, MaxMapY, MapYsize);
 println ("(X:"+Xshift+" Y:"+Yshift+" Ycoord:"+Ycoord+" Yindx:"+Yindx+ ") *** "); // debug
for (Xindx = 0; Xindx < MapXsizeR; Xindx++)
{
CellValue=RxBuff[Xindx+HeadLen+5] & 0x0F; // lower nibble
Xcoord = RealCoord(Xindx*2, Xshift, MinMapX, MaxMapX, MapXsize);
 Objects(Xcoord, Ycoord, CellValue);
CellValue=(RxBuff[Xindx+HeadLen+5] & 0xF0) >> 4; // upper nibble
Xcoord = RealCoord(Xindx*2+1, Xshift, MinMapX, MaxMapX, MapXsize);
Objects(Xcoord, Ycoord, CellValue);
}
}
```

R

- r

Software firmware version request

- TX:

  CmdLen = 1 Params 0

  Example:

```
TxData(0, 'R', 0, 1);
```

- RX:

  CmdLen = 24 Params 25 (string "Ver")

  Example:

```
if (RxData('R',26))
{
      for (i=HeadLen; i < 26+HeadLen; i++)
      {
            Ver[i-HeadLen]= (char)(RxBuff[i]);
      }
}
```

e

- r

Read the code of the last error occurred and reset error condition

- TX:

  CmdLen = 1 Params 0

  Example:

```
TxData(0, 'e', 0, 1);
```

- RX:

  CmdLen = 3 Params 2 (1 int = 2 byte. Most Significant First)

  Example:

```
if (RxData('e',2))
{
      ErrorCode = ((RxBuff[HeadLen] << 8) + (RxBuff[HeadLen+1]));
}
```

f

- w

set "Console Debug" mode

JUST FOR DEBUG

The dsNav firmware acts as a loopback, sending back to the console some values proportional to other received values, in order to check communication and right decoding and encoding of received and transmitted packets.

- TX:

  CmdLen = 1 Params 0

  Example:

```
TxData(0, 'f', 0, 1);
```

- RX:

  N/A

s

- w

Scheduler parameters setting

Read detailed explanations on the manual for the meaning of every single value

- TX:

  CmdLen = 1 Params 0

  Example:

```
for (i=0; i<64; i++)
{
   TxIntValue[i] = Seq[i];
}
TxData(0, 's', 64, 1);
```

- RX:

  N/A

z

- r

the dsNav sends back a text string, for a bidirectional debug of the communication chain

- TX:

  it just needs HEADER + character ‘z’. Just type @z on a simple terminal program (e.g. Hyperterminal)

- RX:

  CmdLen = 25, the test string that will be displayed on terminal console

--- Navigation settings commands

D

- w

Setting reference coordinates X, Y computing distance (Mode C)
The robot navigates for N mm starting from current coordinates and with the current orientation

- TX:

  CmdLen = 3 Params 2 (1 int = 2 byte. Most Significant First)
  Example:

```
TxIntValue[0] = DesDist;
TxData(0, 'D', 1, 1);
```

- RX:

  N/A

P

- w

Setting reference coordinates X, Y in mm (Mode B)
The robot navigates toward the (X, Y) point

- TX:

  CmdLen = 5 Params 4 (2 int = 4byte. Most Significant First)
  Example:

```
TxIntValue[0] = DesCoordX;
TxIntValue[1] = DesCoordY;
TxData(0, 'P', 2, 1);
```

- RX:

  N/A

O

- w

Sets the reference orientation angle in degrees (absolute)

- TX:

  CmdLen = 3 Params 2 (1 int = 2 byte. int Most Significant First)
  Example:

```
TxIntValue[0] = (int)(DesAngle);
TxData(0, 'O', 1, 1);
```

- RX:

  N/A

o

- w

Sets the reference orientation angle in degrees as a delta of the current Theta (relative)

- TX:

  CmdLen = 3 Params 2 (1 int = 2 byte. Most Significant First)
  Example:

```
TxIntValue[0] = (int)(DeltaAngle);
TxData(0, 'o', 1, 1);
```

- RX:

  N/A

S

- w

Reference speed setting in mm/s

The robot travels at this speed unless other events occur

- TX:

  CmdLen = 3 Params 2 (1 int = 2 byte. Most Significant First) range -999 +999

  Example:

```
TxIntValue[0] = DesSpeed;
TxData(0, 'S', 1, 1);
```

- RX:

  N/A

H

- w

Emergency. Immediate Halt of both motors without decelerating ramp

- TX:

  CmdLen = 1 Params 0

  Example:

```
TxData(0, 'H', 0, 1);
```

- RX:

  N/A

--- Navigation read commands

A

- r

All mean parameters request:

Data

0 – 1 Vmean: mean speed. The speed of the robot obtained averaging the speed of the two wheels
2 – 3 Ctotal: the sum of the current used by both motors
4 – 5 PosX,
6 – 7 PosY: the position of the robot as the X, Y coordinates estimated by dsNav by odometry
8 – 9 Theta: orientation angle of the robot in its own reference system
10 IdlePerc: idle time of the MCU on dsNav in percentage: work load of the processor

- TX:

CmdLen = 1 Params 0

Example:

```
TxData(0, 'A', 0, 3);
```

- RX:

CmdLen = 12 Params 11 (5 int = 10 byte. Most Significant First + 1 byte)

Example:

```
if (RxData('A',13))
{
 MesSpeed = ((RxBuff[HeadLen] << 8(RxBuff[HeadLen+1]));
 Current = (float)((RxBuff[HeadLen+2] << 8) + (RxBuff[HeadLen+3]));
 Xpos = ((RxBuff[HeadLen+4] << 8) + (RxBuff[HeadLen+5]));
 Ypos = ((RxBuff[HeadLen+6] << 8) + (RxBuff[HeadLen+7]));
 MesAngle = ((RxBuff[HeadLen+8] << 8) + (RxBuff[HeadLen+9]));
 CompassAngle = ((RxBuff[HeadLen+10] << 8) + (RxBuff[HeadLen+11])) / 10;
 IdlePerc = RxBuff[HeadLen+10];
}
```

a

- r

All detailed parameters request:

Data

0 – 1 Vr: the speed of the right wheel
2 – 3 Vl: the speed of the left wheel
4 – 5 Cr: the current used by right motor
6 – 7 Cl: the current used by the left motor
8 – 9 PulseR: the encoder pulses counted by right encoder since last measure
10 – 11 PulseL: the encoder pulses counted by left encoder since last measure

- TX:

CmdLen = 1 Params 0

Example:

```
TxData(0, 'a', 0, 3);
```

- RX:

CmdLen = 13 Params 12 (6 int Most Significant First)

Example:

```
if (RxData('a',12))
{
 SpeedR = (((RxBuff[HeadLen] << 8) + (RxBuff[HeadLen+1])));
 SpeedL = (((RxBuff[HeadLen+2] << 8) + (RxBuff[HeadLen+3])));
 CurrentR = (float)((RxBuff[HeadLen+4] << 8) + (RxBuff[HeadLen+5]));
 CurrentL = (float)((RxBuff[HeadLen+6] << 8) + (RxBuff[HeadLen+7]));
 PulseR = (((RxBuff[HeadLen+8] << 8) + (RxBuff[HeadLen+9])));
 PulseL = (((RxBuff[HeadLen+10] << 8) + (RxBuff[HeadLen+11])));
}
```

--- Constant parameters setting commands

J

- w

Settings PID coefficients for DistPid: DistKP, DistKI, DistKD

Data

0 – 1 DistKp

2 – 3 DistKi

4 – 5 DistKd

- TX:

CmdLen = 7 Params 6 (3 int MSF) = (Kx * 10000) range 0 9999

```
DistKp = InputDistKp.getValue();
DistKi = InputDistKi.getValue();
DistKd = InputDistKd.getValue();
TxIntValue[0] = DistKp;
TxIntValue[1] = DistKi;
TxIntValue[2] = DistKd;
TxData(0, 'J', 3, 1);
```

- RX:

N/A

K

- w

Settings PID coefficients for SpeedPID:

Data

0 – 1 Kp right

2 – 3 Ki right

4 – 5 Kd right

6 – 7 Kp left

8 – 9 Ki left

10 – 11 Kd left

- TX:

CmdLen = 13 Params 12 (6 int MSF) = (Kx * 10000) range 0 9999

Example:

```
KpR = InputDistKpR.getValue();
KiR = InputDistKiR.getValue();
KdR = InputDistKdR.getValue();
KpL = InputDistKpL.getValue();
KiL = InputDistKiL.getValue();
KdL = InputDistKdL.getValue();
TxIntValue[0] = KpR;
TxIntValue[1] = KiR;
TxIntValue[2] = KdR;
TxIntValue[3] = KpL;
TxIntValue[4] = KiL;
TxIntValue[5] = KdL:;
TxData(0, 'K', 6, 1);
```

- RX:

N/A

k

- w

Settings PID coefficients for AnglePid: AngleKP, AngleKI, AngleKD

- TX:

  CmdLen = 7 Params 6 (3 int MSF) = (Kx * 10000) range 0 9999

  Example:

```
AngleKp = InputAngleKp.getValue();
AngleKi = InputAngleKi.getValue();
AngleKd = InputAngleKd.getValue();
TxIntValue[0] = AngleKp;
TxIntValue[1] = AngleKi;
TxIntValue[2] = AngleKd;
TxData(0, 'k', 3, 1);
```

- RX:

  N/A

L

- w

Speed constant parameters designation: KvelR, KvelL

Read detailed explanations on how to compute these values

- TX:

  CmdLen = 9 Params 8 (2 long MSF) = KvelX

  Example:

```
K = InputAxle.getValue();
Wheel1 = InputWheel1.getValue();
Wheel2 = InputWheel2.getValue();
Cpr = InputCpr.getValue();
Gear = InputGear.getValue();

Kvel1=(Wheel1*PI*Fcy*32768) /(Cpr*Gear*1000) ;
Kvel2=(Wheel2*PI*Fcy*32768) /(Cpr*Gear*1000) ;
TxIntValue[0] = (int)(Kvel1);
TxIntValue[1] = (int)(Kvel2);
TxData(0, 'L', 2, 2);
```

- RX:

  N/A

M

- w

Mechanical costants:

Axle size: the length of the axle as the distance between the center of right and left wheels

KspR, KspL: the space traveled by the wheel for every single encoder tick (Right and Left)

- TX:

  CmdLen = 13 Params 12 (3 long MSF)

  Example:

```
K = InputAxle.getValue();
Wheel1 = InputWheel1.getValue();
Wheel2 = InputWheel2.getValue();
Cpr = InputCpr.getValue();
Gear = InputGear.getValue();

Ksp1 = (long)((Wheel1*PI*1000)/(Cpr*Gear*4));
Ksp2 = (long)((Wheel2*PI*1000)/(Cpr*Gear*4));
TxIntValue[0] = K/100;
TxIntValue[1] = (int)(Ksp1);
TxIntValue[2] = (int)(Ksp2);
TxData(0, 'M', 3, 2);
```

- RX:

  N/A

--- Sensors commands

d

- w

Distance from objects and target readings on three sides coming from sensors board

- TX:

CmdLen = 9 , Params 8 (6 bytes + 1 int)

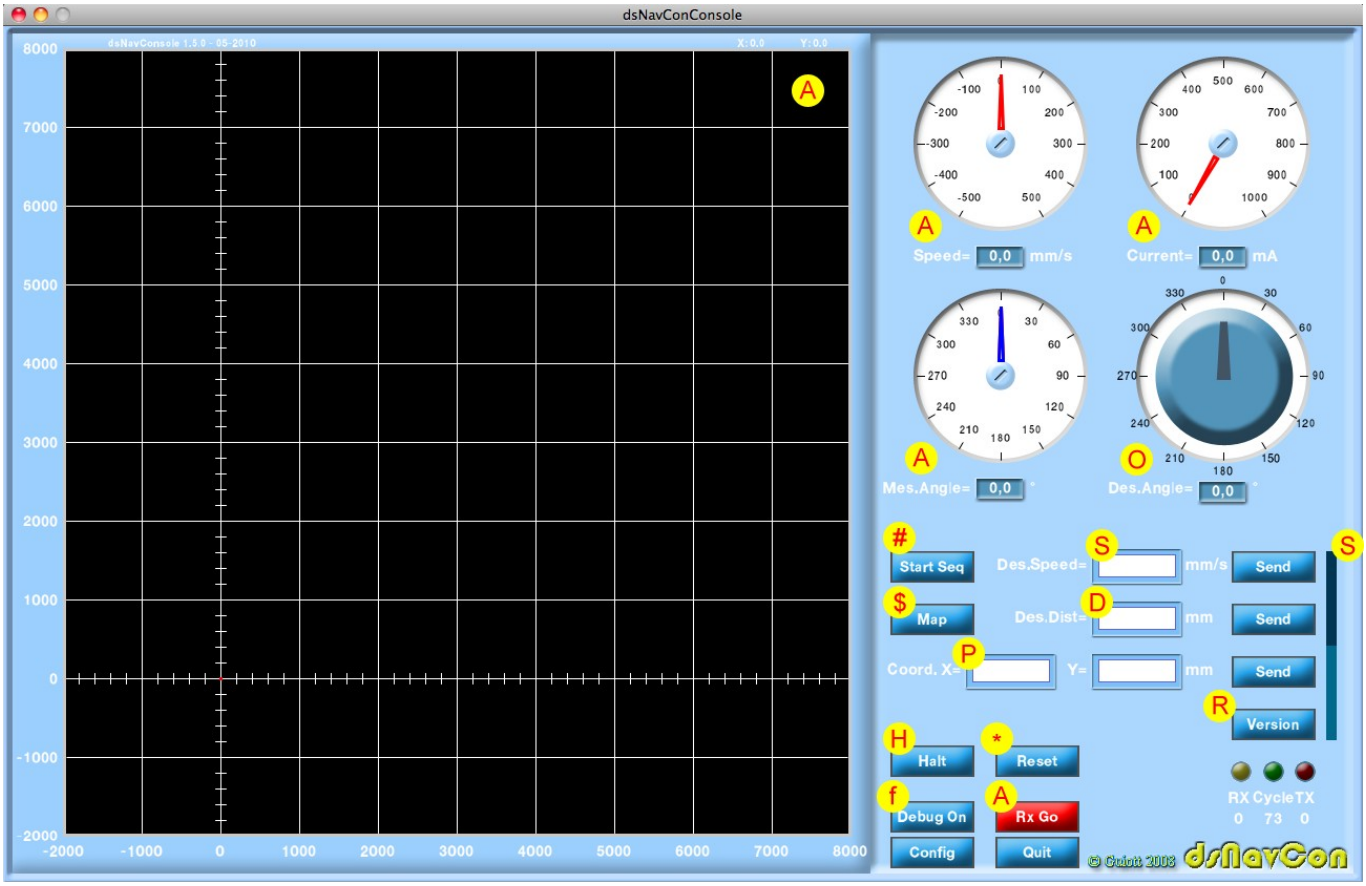
Example in Arduino language:

```
void DataPrint(void)
{// send read data on serial port to Navigation Control Board
  int i;                        // generic index
  byte TxBuff[16];
  byte ChkSum=0;
  TxBuff[0] = 64;               // Header = "@"
  TxBuff[1] = 0;                // Id = 0
  TxBuff[2] = 100;              // command = "d"
  TxBuff[3] = 9;                // command length
  TxBuff[4] = Dist[L];          // data 0
  TxBuff[5] = Dist[C];          // data 1
  TxBuff[6] = Dist[R];          // data 2
  TxBuff[7] = Target(L);        // data 3
  TxBuff[8] = Target(C);        // data 4
  TxBuff[9] = Target(R);        // data 5
  TxBuff[10]= (byte)(CmpBearing >> 8);// data 6
  TxBuff[11]= (byte)(CmpBearing); // data 7 (data 6 + 7 = int)
  for (i=0; i<=11; i++)
  {
    ChkSum += TxBuff[i];
    Serial.print(TxBuff[i], BYTE);
  }
  Serial.print(ChkSum);
}
```

- RX:

N/A

Commands as used on console and console description.



Main Panel.
This is the first panel that appears when the console program is launched and it's the mostly used.
The letters inside the circle indicates the command as previously described.

A All mean parameters request
After pressing "Rx Go" button, and only if the PC communication port is correctly configured, the command is sent periodically from console program to dsNav board at every blink of the green led, with cycle time shown below it (in ms). Pressing again this button the cycle stops.
The dsNav replays with the requested values:
Vmean: mean speed. Shown on "Speed" gauge
Ctotal: current shown "Current" gauge
PosX, PosY: shown as a graph
Theta: estimated orientation angle. Shown on "Mes.Angle" gauge with red needle. The blu needle optionally shows the angle as measured by digital compass and received by optional sensors board.
IdlePerc: idle time of the MCU. Shown as a percentage under the yellow led.

O Sets the reference orientation angle in degrees (absolute)
As the desired angle varies on rotating the knob, this new value is sent to the dsNav.

Start scheduler sequence
Pushing this button starts the sequence of actions previously sent to the dsNav with "Sequencer" panel.

$ One row of map grid sending request
Requests the values for cells of the field as mapped traveling: obstacles, targets or simply the presence on that cell, and display that values on the map.

S Reference speed setting in mm/s
The desired traveling speed for the robot may be sent writing down the value (in mm/s) in the input field or moving with the mouse the slider. This value can be both positive or negative, meaning traveling backward. The robot approaches the desired speed with an accelerating (or decelerating) ramp and maintains that speed unless an obstacle is found if optional sensors board is present.

D Setting reference coordinates X, Y computing distance (Mode C)
The robot navigates for the distance written in the input field (in mm) starting from current coordinates and with the current orientation.

P Setting reference coordinates X, Y in mm (Mode B)
The robot navigates toward the point with coordinates written in these input fields.

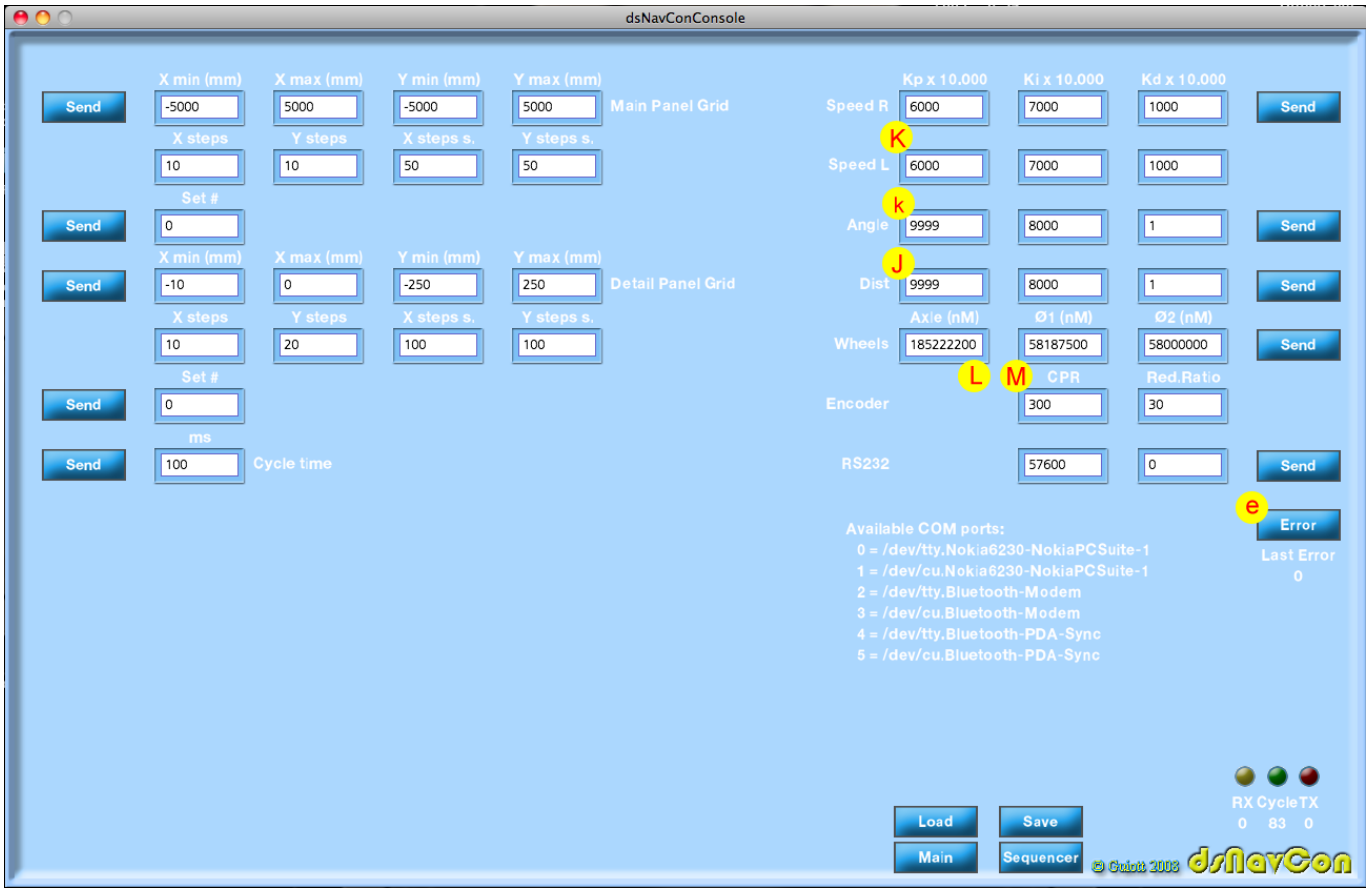
R Software firmware version request
Pushing this button the command is sent and dsNav replays with a string containing its firmware version information. This string is displayed besides the button.

H Immediate Halt of both motors without decelerating ramp
This is useful for emergency stopping the robot immediately. It is not healthy for the motors and gears, especially for heavy robots. Use it carefully.

***** Board software reset
Useful to remotely restarts the dsNav. For security reason, this command has to be sent three times consecutively in order to perform reset. After the reset the robot sets a new reference system using the current position and orientation.

f set "Console Debug" mode
JUST FOR DEBUG, in order to test the communication between console and dsNav and right decoding and encoding of received and transmitted packets, also if motors and encoders are not connected.
The dsNav firmware acts as a loopback, sending back to the console some values proportional to other received values.

The “Config” button allows the switch to the configuration panel. The “Quit” button stops the program.
The RX yellow led flashes at every packet received from the dsNav.
The Cycle green led flashes when an automatic parameters request is periodically sent to dsNav.
The TX red led flashes when commands are manually sent with buttons or knob.
The number below this led shows how many communication errors occurred since the program was launched. A description of the last error occurred is displayed to the left.



Configuration Panel.
This panel is used to configure the parameters on both the robot and console according to the procedures described on manual.

K Settings KP, Ki, Kd PID coefficients for the procedure on dsNav board that takes care of the speed of left and right motors.

k Settings PID coefficients for AnglePid: AngleKP, AngleKI, AngleKD.
This procedure maintains the robot oriented to the desired angle.

J Settings PID coefficients for DistPid: DistKP, DistKI, DistKD
This procedure allows the robot approaching the target with the best precision and speed.

L Speed constant parameters designation: KvelR, KvelL
M Mechanical constants
Writing down in these fields the physical dimensions of the robot, the console computes and send the parameters as requested by dsNav.
Axle size: the length of the axle as the distance between the center of right and left wheels
Ø1 and Ø2: the diameters of the wheels.
Since the odometry is affected by cumulative error, both these parameters are expressed in nM (nano meters) to have the best precision on position estimation.
Encoders CPR: Count Per Revolution.
Red. Ratio: gear reduction ratio. How many revolutions the encoder does for each revolution of the wheel.

e Reads and displays the code of the last error occurred and reset error condition on dsNav.

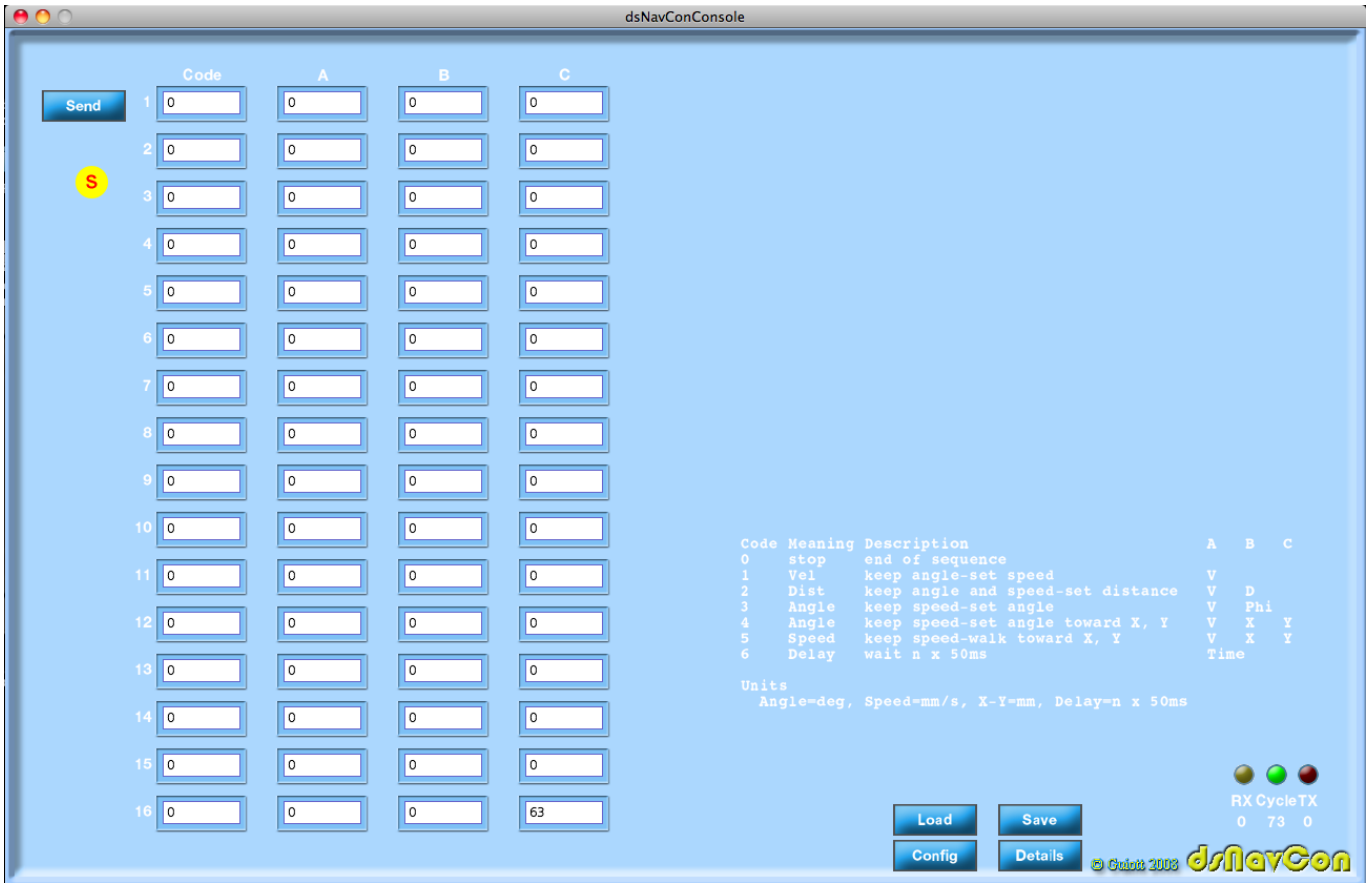
A list of the available COMM ports on PC is displayed in this panel. The readout may vary for different operating systems. The port is open on setting its number in the relative field. Only valid numbers are accepted. The COMM speed is set as well with this button.

The fields and buttons on the left allow the configuration of the chart for main and detail panels, setting grids and limits.
A number of predefined settings are available for both panels.

The cycling time (in ms) for automatic parameters request can be set with the relative button.

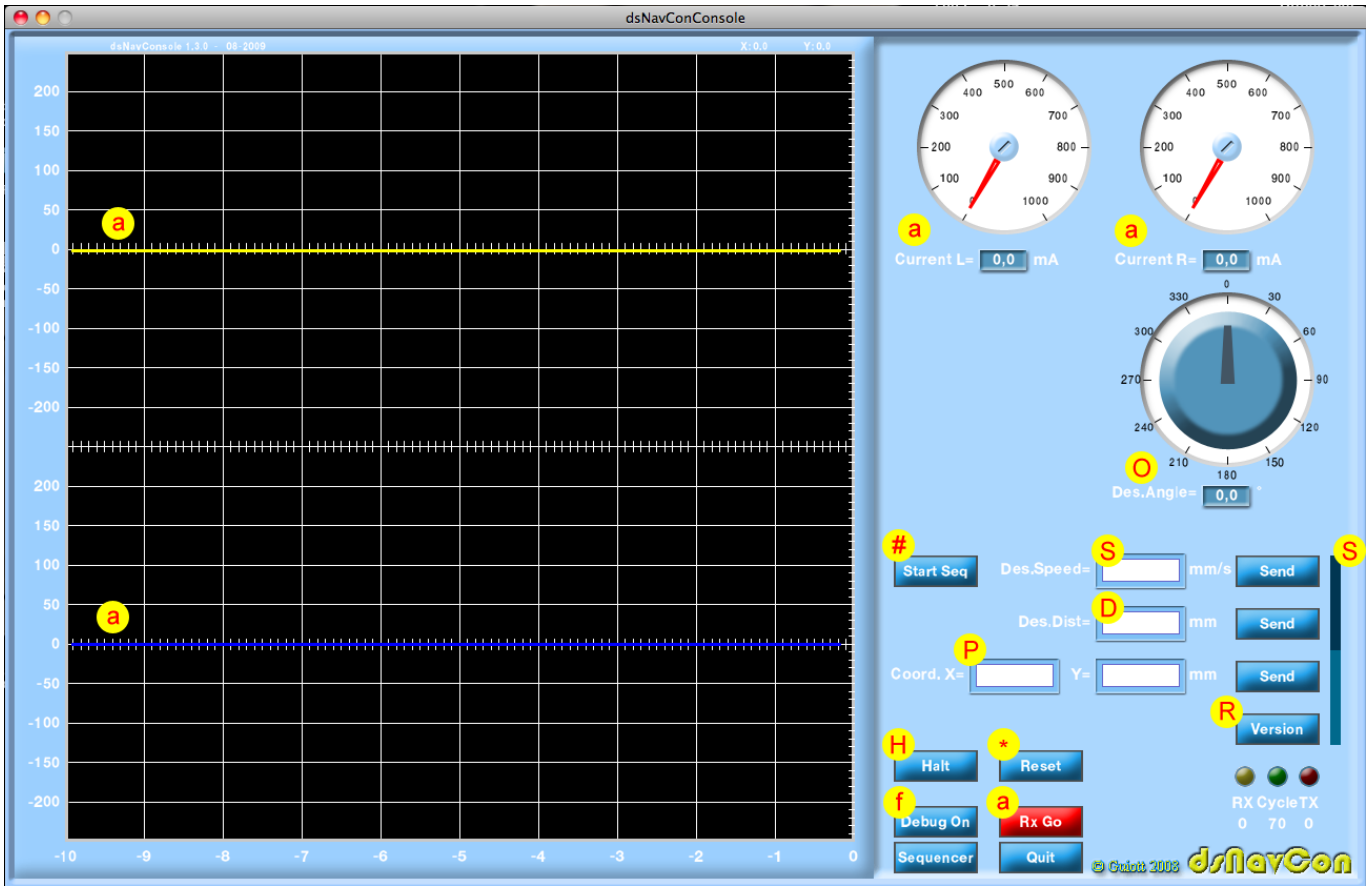
The entire configuration can be saved to a file on the PC and restored with “Load” and “Save” buttons.

The “Main” and “Sequencer” buttons switch back and forth between panels.



Sequencer Panel.

s Scheduler parameters setting
A 16 actions sequence list can be configured and permanently stored on dsNav flash ram with this panel.
The syntax for the actions to execute is displayed here as well.
Sending the # (Start scheduler sequence) command to dsNav, the robot starts performing the required actions in sequence.
This could be useful to follow repetitive paths as needed, for example, for an UMBmark calibration.



Details Panel.
This one is similar to “Main” panel.
Whit ***a*** command (all detailed parameters request) it displays the values for current and speed relative to each single motor instead on mean computed values.
The graphs show the trend of the speed for right and left wheels, this is very useful to calibrate the Kp, Ki, Kd parameters for the specific robot.