Introduction Tutorial

Setting up the Robonova

The Robonova robot comes pre-assembled and ready to walk. The DASL kit contains:

- Software CD
- Manual
- Battery charger
- Serial computer interface cable (DB9-mini)
- Remote control & IR receiver (on Robonova's head)
- High-speed serial adapter and serial level shifter
- Sensors (to be added)

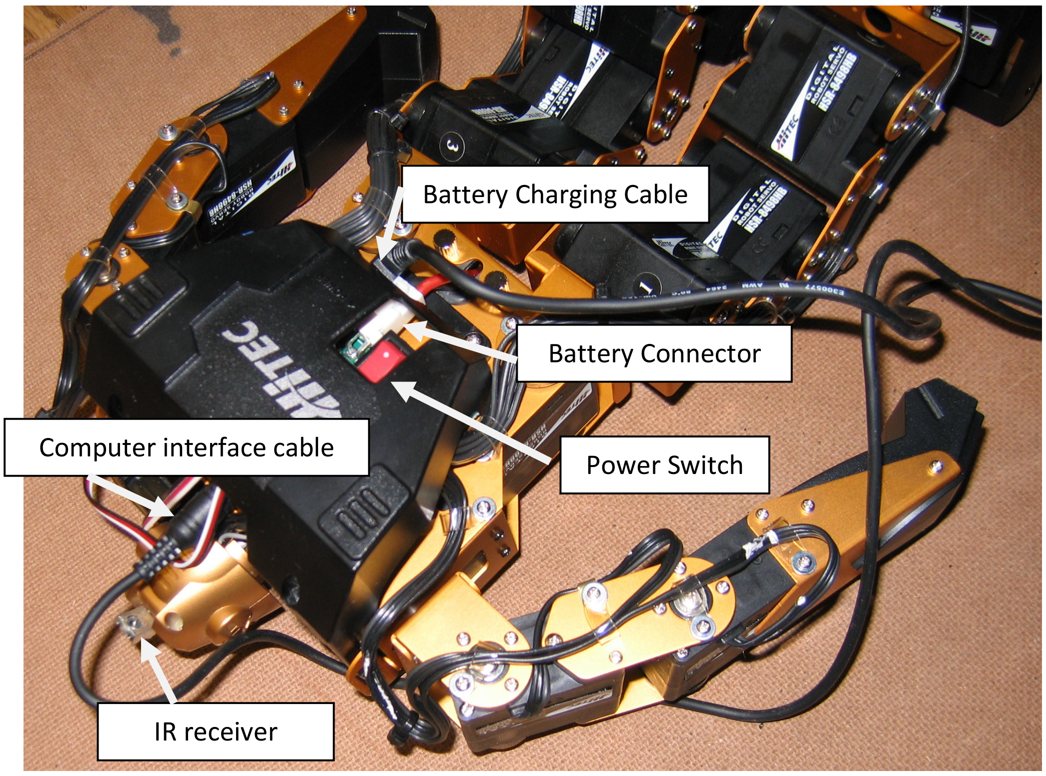
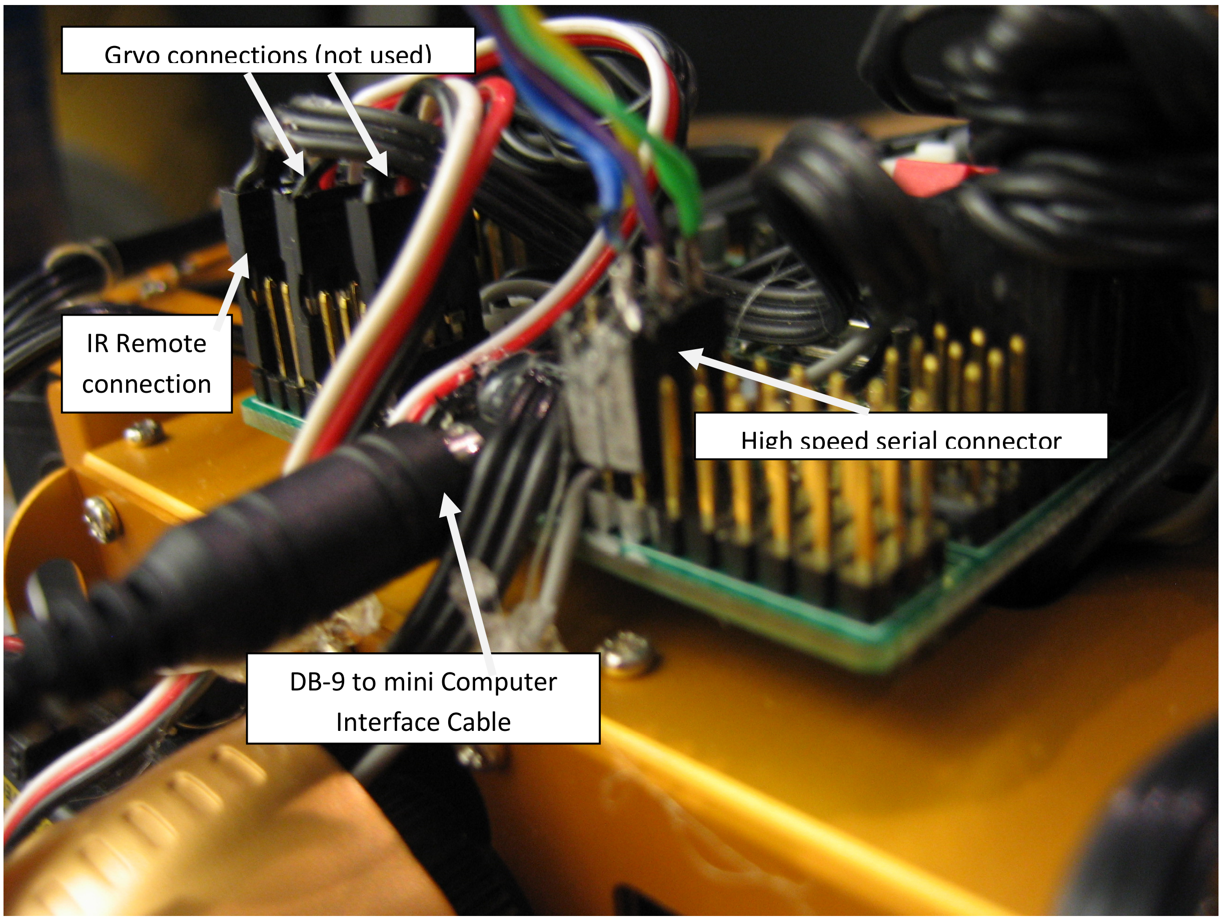


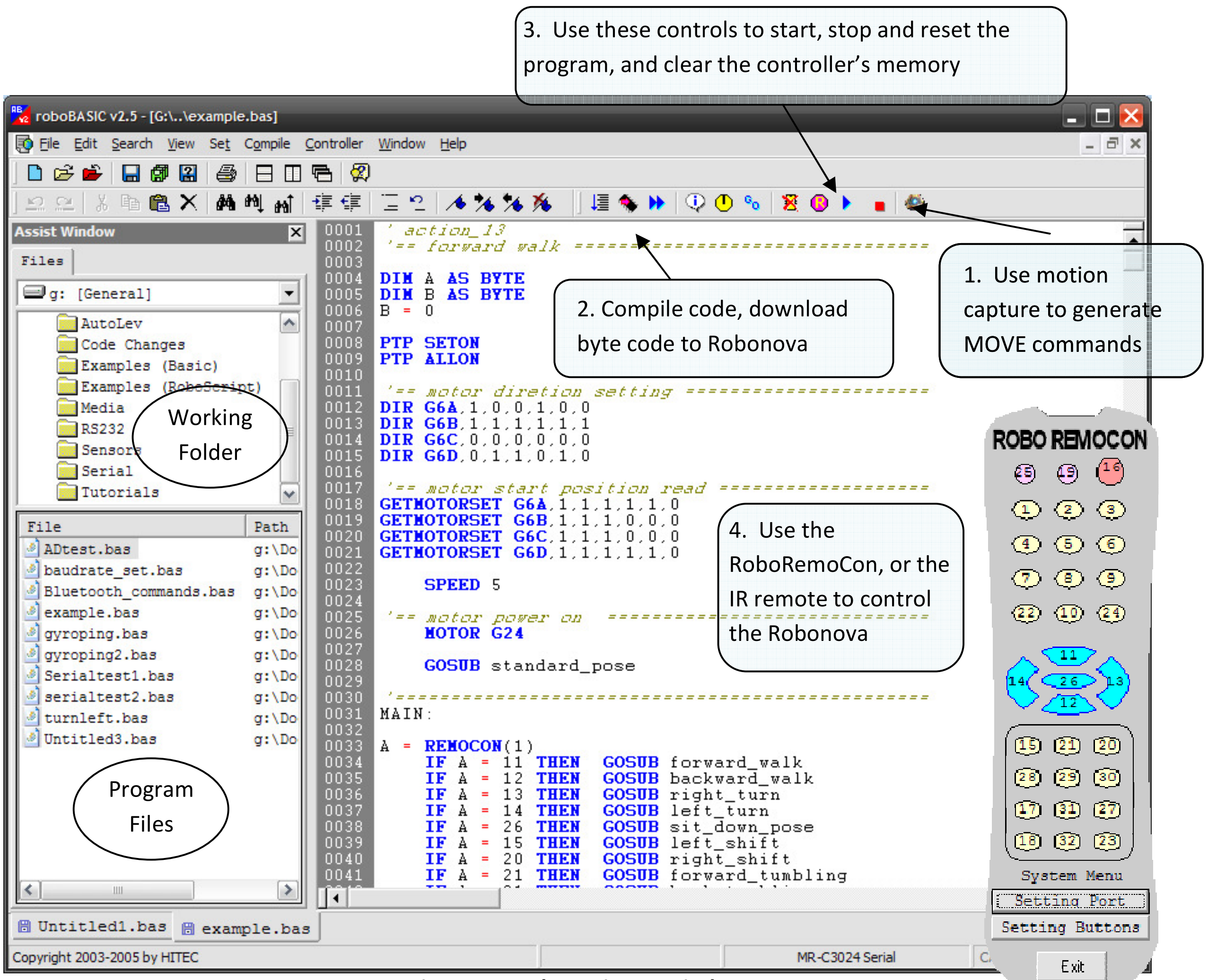
Figure 1 - Robonova connections

The power for the Robonova comes from its external supply and a 6V NiMH battery pack. While running experiments and debugging programs, keep the power connector in the charge socket to lengthen the run time. From a discharged battery, it typically takes about 2 hours to fully recharge.


Grvo connections (not used)
IR Remote connection
High speed serial connector
DB-9 to mini Computer Interface Cable
Figure 2 - Microcontroller & wire connections

Once the Robonova is powered, connect it to the host computer using the DB9 to mini jack serial cable (Figure 2). This cable allows the computer to issue commands and flash the MR-C3024 microcontroller's ROM with your program.

Download and install the latest version of RoboBasic (2.5.61, included on the tutorial website). Also download and install the sample code files included with this tutorial. Open the program window, and load "example.bas". You should see the program window as in Figure 3.



```
' action_13
'== forward walk ==========================================

DIM A AS BYTE
DIM B AS BYTE
B = 0

PTP SETON
PTP ALLON

'== motor diretion setting ======================

DIR G6A,1,0,0,1,0,0
DIR G6B,1,1,1,1,1,1
DIR G6C,0,0,0,0,0,0
DIR G6D,0,1,1,0,1,0

'== motor start position read ===================
GETMOTORSET G6A,1,1,1,1,1,0
GETMOTORSET G6B,1,1,1,0,0,0
GETMOTORSET G6C,1,1,1,0,0,0
GETMOTORSET G6D,1,1,1,1,1,0

    SPEED 5

'== motor power on  =============================
    MOTOR G24

    GOSUB standard_pose

'===============================================
MAIN:

A = REMOCON(1)
    IF A = 11 THEN  GOSUB forward_walk
    IF A = 12 THEN  GOSUB backward_walk
    IF A = 13 THEN  GOSUB right_turn
    IF A = 14 THEN  GOSUB left_turn
    IF A = 26 THEN  GOSUB sit_down_pose
    IF A = 15 THEN  GOSUB left_shift
    IF A = 20 THEN  GOSUB right_shift
    IF A = 21 THEN  GOSUB forward_tumbling
```

Figure 3 – RoboBasic IDE window

With the Robonova plugged in, download and run example.bas. Start the program "RoboRemoCon" (included with the robobasic installation), and press the numbered buttons to execute various commands. Notice how the buttons are numbered, and how each number corresponds to a subroutine in the example code.

In general, the program structure will consist of 3 parts:

1. Initialization
    a. Power on the motors with MOTOR G24
    b. Set motor directions (typically default directions) with the DIR command
    c. Read motor positions (MOTORSET)
    d. Set initial speed with SPEED command
    e. Set PTP on or off (depending on the application)
2. MAIN loop (simple)
    a. Read a source of motion commands, either the remote control or the serial connection

    b. Parse the command input (either ASCII characters from serial, or a button number from the remote)
    c. Use IF statements or other choice structure to execute the specified command
3. Subroutines
    a. Subroutines are like a bookmark in the code. The command GOSUB *examplesub* simply jumps to the code beginning at "*examplesub*:", and continues executing.
    b. MOVE commands are typically executed in subroutines, which basically tell the robot to seek to the motor positions, at a speed set by the SPEED command. The example code to see how the motions are represented by a series of poses.
    c. Always end a subroutine with RETURN or END, which tells the program to go back to where the sub was originally called, or end the program. If the program doesn't get either of these commands, it will simply execute the next line of code, which could be another subroutine, and almost surely not what you want to do.

Note that the command *byte*=REMOCON(1) returns the value of the first button pushed within a time window of about 200 ms. This process also halts execution of other code while it's waiting, so make sure to probe the remote only after critical functions are done.

Motion control window

Open the motion control window by pressing the far left button on the toolbar:
When the Robonova is connected and powered on, this windows lets you interactively position the motors (Figure 4).

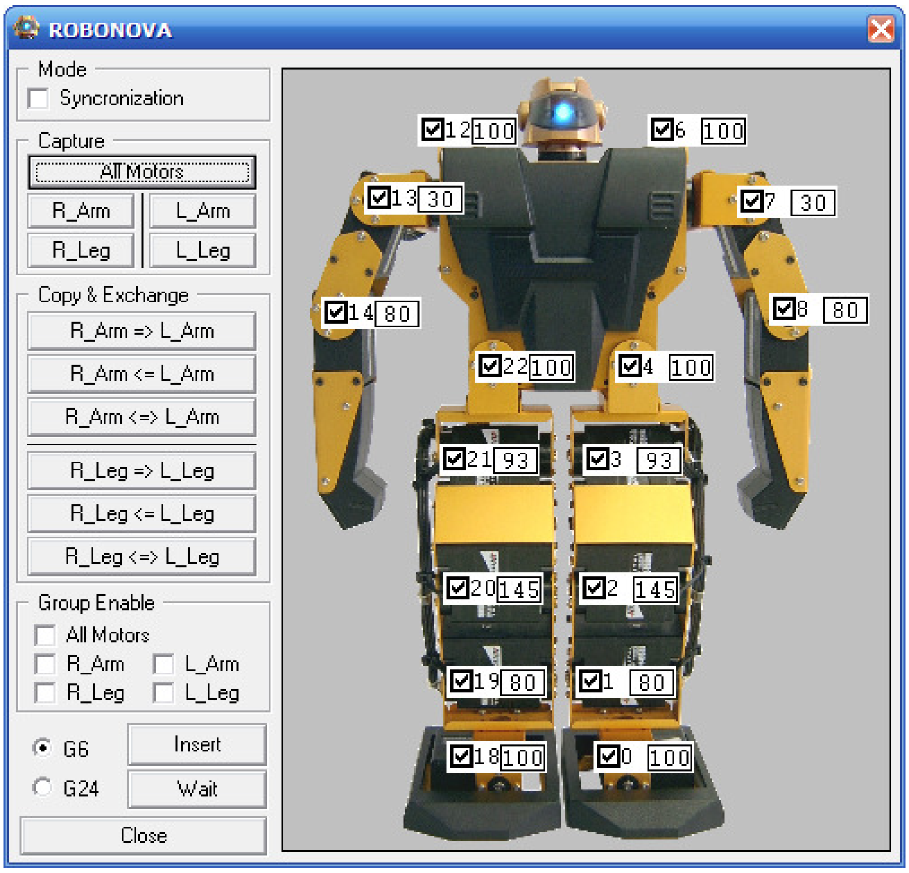


Figure 4 - Motor position control

Hover the mouse over one of the joints, and a slider/play button set will appear. Push the single arrow to move the motor by one degree, or the double arrow to move it by 5 degrees. The slide lets you freely and quickly position the motors. To capture a particular position as a MOVE command, press the insert button, and it will create a MOVE24 command in your code to move the robot to the current position. Doing this repeatedly will create a series of motions for the robot to follow.

For making repetitive motions, the copy & exchange buttons let you duplicate or exchange the positions of the arms and legs. This way, you only need to pose one side of the body for a symmetrical motion.

Using the blank program template and the motion capture system, make the Robonova:

- Bow
- Wave hello
- Crouch & touch its toes from the side

Design tips for posed motion

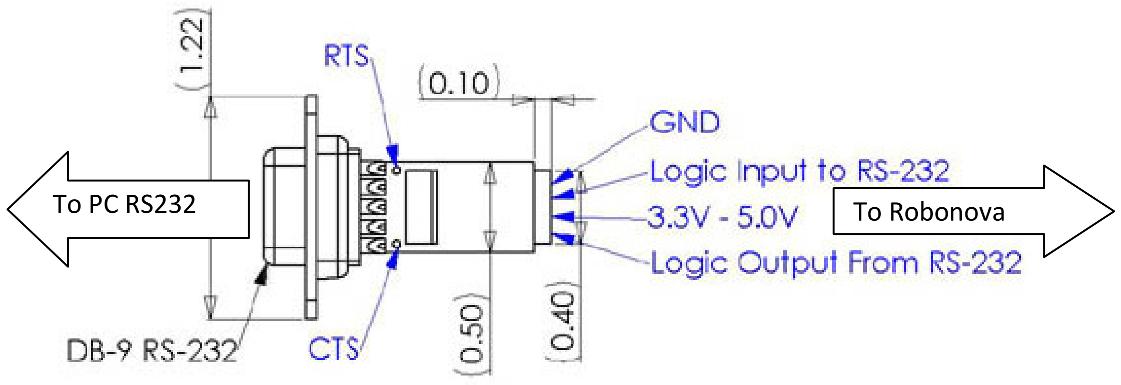
- Keep the center of mass over the feet at all times, or the robot will tip over.
- Stability of a given pose can be increased by widening the stance, but changing the foot placement requires careful planning, as simply sliding the foot can be difficult. Example code is included for a sidestep that lifts the foot off the ground. Can you improve it?

The Robonova has a lot of trouble turning, since it can't rotate the hip joint in the Z axis. The left and right turn examples work by placing the feet at an angle and bringing them together again, hopefully rotating the body. Load and run the turning example code to see its limitations. By modifying the code to lean the robot forward slightly, it transfers the weight to the toes, which will slip more freely during the turn, allowing the robonova to turn more smoothly. Another way to improve the turning behavior is to give it "dance shoes". This works by placing a high-friction surface on the heel such as rough grip tape, and a low friction surface at the toe, like plastic.

Load and run the example code again and make the Robonova walk forward and backwards. Notice how the body sways with each transition. One way to reduce this sway is to move the arms in the opposite direction of the legs, much like humans do. Try to tune the arm wing to cancel some of the wobble in the forward walk.

Serial Communication

The Robonova can send and receive data through its high speed serial UART. The pins labeled ETX and ERX are TTL (5V) level serial pins. Connect the supplied cable to the serial pins and the serial level shifter. Plug the DB-9 side of the serial level shifter



On the computer side, open a hyperterminal window, and create a new connection. Set the COM port to whatever port is connected to the ETX/ERX pins. Choose these settings:
- 57600 baud
- 1 start bit
- 1 stop bit
- No parity

- No flow control

To send a byte over the serial port to the robonova, simply type it in the hyperterminal window. The characters you type do not appear in the window unless you select “echo characters in terminal window” in the options menu.

In RoboBasic, the command “ETX 57600, *byte*” sends the *byte* over the serial line at 57600 baud.

“ERX 576900,*byte* ,*retry_subroutine”* receives a byte from the serial line and stores it in *byte*. The retry_subroutine is what the command will execute if it can’t find any serial input.

An example transmit subroutine looks like:

```
DIM B AS BYTE
MAIN:
        B=”B”
        GOSUB serial_transmit
        GOSUB MAIN

serial_transmit:
        ‘B must be a byte variable
        ETX 57600,B
        RETURN
```

Using the above commands, write a program to read characters from the serial input and send the same character back. This is called a loopback test, and determines if your serial hardware and code is functioning correctly.

Bonus: Modify the example code to accept input from the serial line. (i.e. send an “w” character over serial to make it walk forward, or “a” to turn left, etc.