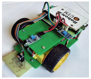
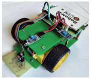


Real-time Programming on RoboKar Using MicroC/OS-II Kernel

Rosbi bin Mamat

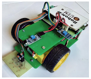
`bmrosbi@gmail.com`

Contents



1. Objectives
2. Introduction to RoboKar
   - RoboKar Physical Construction
   - RoboKar's Brain
   - Actuators on RoboKar
   - Sensors on RoboKar
3. HAL for RoboKar
4. Installing Development Tools
5. Software Development Steps
6. RoboKar Code Template

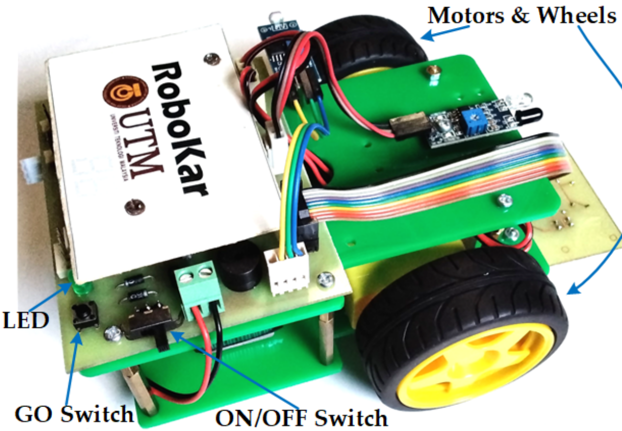
1. Objectives

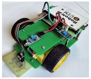


- To introduce the RoboKar mobile robot: the physical hardware, the embedded processor (the brain), on-board sensors & actuators.
- To show how the Hardware Abstraction Layer (HAL) can be used for programming RoboKar.
- To demonstrate how to use GNU dev. tools & MicroC/OS-II kernel for developing real-time software on RoboKar.

2. Introduction to RoboKar

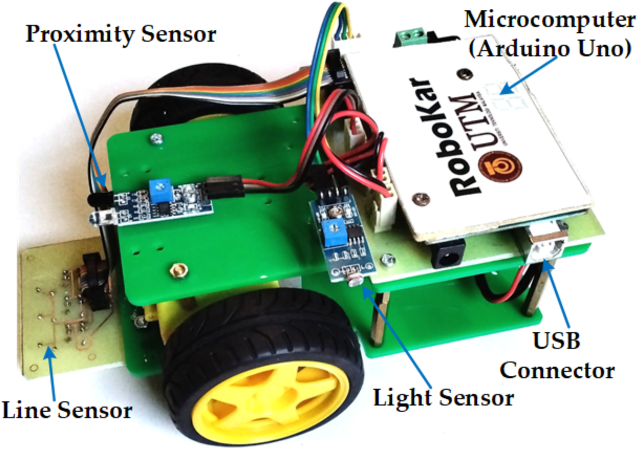
- RoboKar Physical Construction.



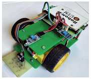


2. Introduction to RoboKar

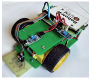
- RoboKar Physical Construction.



2. Introduction to RoboKar

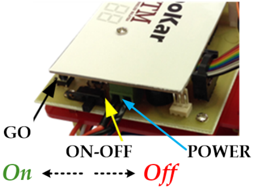


- **RoboKar's Brain** – The brain of RoboKar is Arduino Uno microcomputer based on ATMEL ATmega328P 8-bit Microcontroller (μC). Sofware written for RoboKar is executed by the μC.
- Contains on-chip resources (ROM,RAM, I/O) typically used in embedded applications.
  - **32Kb of Flash ROM.**
  - **2Kb internal RAM.**
- μC may not requires extra off-chip resources to function ⇒ less chips required & lower power requirements.
- μC **Disadvantage**: Difficult/impossible to expand internal memories & I/O.

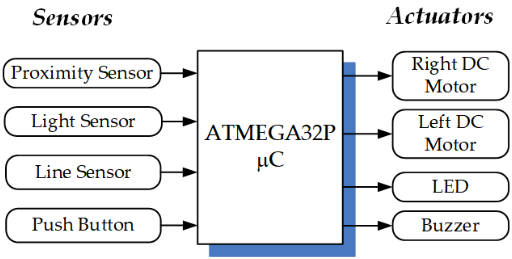


2. Introduction to RoboKar

- **Power supply** is provided by rechargable battery. Must be recharged when low.



- **Sensors & actuators** connected to RoboKar:



Sensors

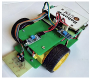
- Proximity Sensor
- Light Sensor
- Line Sensor
- Push Button

ATMEGA32P μC

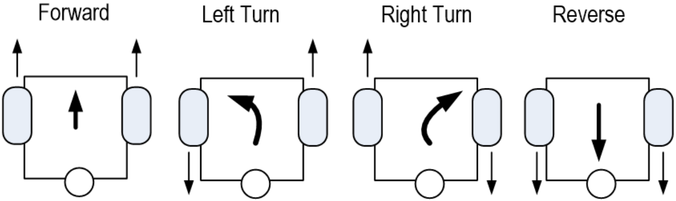
Actuators

- Right DC Motor
- Left DC Motor
- LED
- Buzzer

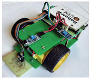
2. Introduction to RoboKar



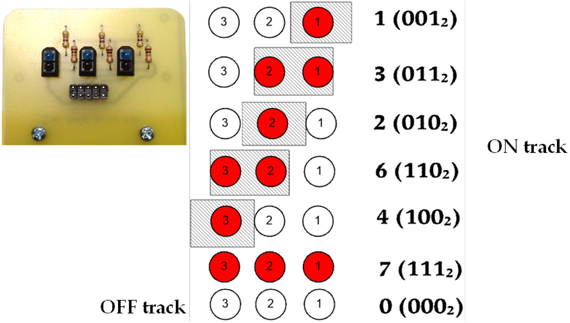
- **Actuators** on RoboKar
  - **LED** & **buzzer** provide visual & audio signals – useful for debugging.
  - **DC motors** are used to rotate the wheels at different speeds, thus move the robot around.
  - 2 DC motors are connected to μC to control the motion of the robot. Code for controlling DC motors is discussed in Section 3.
  - To **navigate** the robot:



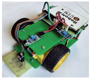
2. Introduction to RoboKar



- **Sensors** on RoboKar
  - **Push button** provides interaction with the user.
  - **Proximity sensor** detects obstacles in front of RoboKar. Any obstacles within 0 – 10 cm will be detected by the sensor.
  - **Light sensor** measures the brightness of light.
  - **Line sensor** informs the position of RoboKar relative to the black line on the white track.



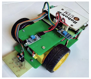
3. HAL for RoboKar



- Writing codes for controlling DC motors, reading sensors values & programming timer for real-time kernel requires deep understanding of how ATmega328 μC h/w works.
- The HAL provides some interface functions for controlling actuators & reading sensors on RoboKar. Thus, user without h/w knowledge can program RoboKar easily.
- To use HAL functions, include the following line in your code & link your code with file `hal_robo.o`.

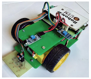
```
#include "..\inc\hal_robo.h"
```

3. HAL for RoboKar



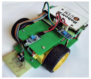
- Functions for **Initializations** on RoboKar:
  - `robo_Setup()` – Initialize **I/O** on RoboKar. Must be called 1st before using the HAL.
  - `OS_ticks_init()` – Initialize ATmega328P μC **internal timer** to produce `10` ms tick interrupt. For use with MicroC/OS-II real-time kernel only.
- Functions for **controlling actuators** on RoboKar:
  - `robo_motorSpeed(lspeed, rspeed)` – Rotates the **robot wheels** with speed `lspeed` (left wheel) & `rspeed` (right wheel). The range for speed values is `-100` – `100`. Negative speed for reverse rotation & positive speed for forward rotation.
  - `robo_Honk()` – Sounds the **buzzer** on RoboKar.
  - `robo_LED_off()`, `robo_LED_on()` & `robo_LED_toggle()` – turn off, on or toggle the **LED** on RoboKar. Useful for debugging.

3. HAL for RoboKar

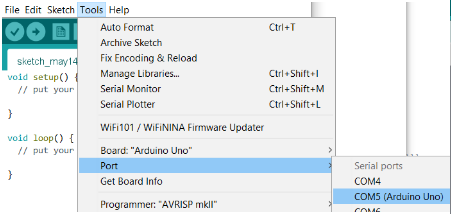


- Functions for **reading sensors** on RoboKar:
  - `robo_proxSensor()` – Read the **proximity sensor** & returns `1` if obstacles is within `0` – `10` cm infront of RoboKar else returns `0` if there is no obstacles.
  - `robo_lightSensor()` – Read the **light sensor** & returns a value between `0` – `100` which represents the percentage of brightness of the light. Examples of readings: 60% (room brightness), less than 30% (very dark), greater than 80% (very bright).
  - `robo_lineSensor()` – Read **line sensor** & returns a value between `0` – `7` which represents the position of RoboKar on the track as shown on Slide 9.
  - `robo_goPressed()` – Check whether the **push button** is pressed. Returns 1 if pressed. Returns 0 if not pressed.
  - `robo_wait4goPress()` – **Wait** for the **push button** to be pressed before continuing with the rest of the program.

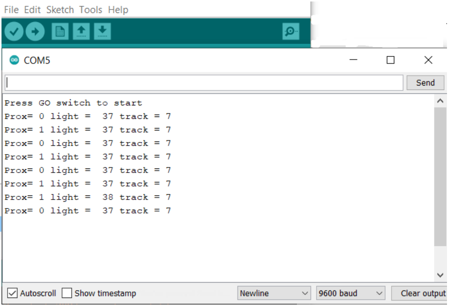
3. HAL for RoboKar



- Misc. functions for displaying on **serial plotter** on Arduino IDE – useful for debugging purposes.
  - `cputchar(unsigned char c)` – similar to standard `putchar()`.
  - `cgetchar()` – similar to standard `getchar()`.
  - `cputs(char *s)` – similar to standard `puts()`.
  - `cprintf(const char *format, ... )` – similar to standard `printf()`. But, limits to integer number display only.


File Edit Sketch Tools Help
sketch_may14
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Serial ports
COM4
COM5 (Arduino Uno)
COM6

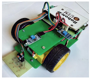
```
void setup() {
  // put your

}

void loop() {
  // put your

}
```



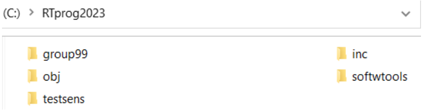
```
Press GO switch to start
Prox= 0 light =  37 track = 7
Prox= 1 light =  37 track = 7
Prox= 0 light =  37 track = 7
Prox= 1 light =  37 track = 7
Prox= 0 light =  37 track = 7
Prox= 1 light =  37 track = 7
Prox= 1 light =  38 track = 7
Prox= 0 light =  37 track = 7
```

☑ Autoscroll □ Show timestamp Newline 9600 baud

4. Installing Development Tools

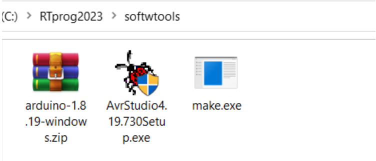


- Free / open-source software dev. tools & MicroC/OS-II real-time kernel will be used.
- Tools used:
  - **Arduino IDE** software – the GNU C Compiler for ATmega328P μC is used.
  - **AvrStudio** – IDE from ATMEL to be used with the GNU C Compiler.
  - **Make.exe** – tool for automating the compilation process used by **AvrStudio**.
  - The provided support files – `includes`, `object` & template files.



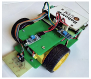
(C:) > RTprog2023

group99
obj
testsens
inc
softwtools



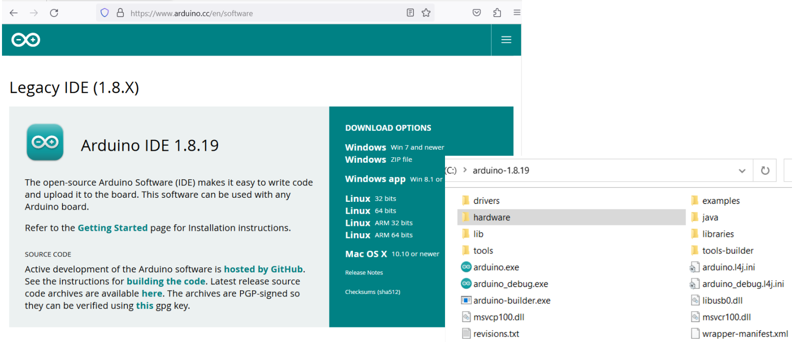
(C:) > RTprog2023 > softwtools

arduino-1.8.19-windows.zip
AvrStudio4.19.730Setup.exe
make.exe



4. Installing Development Tools

- **Install Arduino IDE** – download the file `arduino-1.8.19-windows.zip` from Arduino webpage & just unzip it at `C:\`.



https://www.arduino.cc/en/software

Legacy IDE (1.8.X)

Arduino IDE 1.8.19

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any Arduino board.

Refer to the **Getting Started** page for Installation instructions.

SOURCE CODE

Active development of the Arduino software is **hosted by GitHub**. See the instructions for **building the code**. Latest release source code archives are available **here**. The archives are PGP-signed so they can be verified using **this** gpg key.

DOWNLOAD OPTIONS

Windows Win 7 and newer
Windows ZIP file

Windows app Win 8.1 or

Linux 32 bits
Linux 64 bits
Linux ARM 32 bits
Linux ARM 64 bits

Mac OS X 10.10 or newer

Release Notes

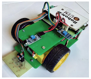
Checksums (sha512)

(C:) › arduino-1.8.19

drivers
hardware
lib
tools
arduino.exe
arduino_debug.exe
arduino-builder.exe
msvcp100.dll
revisions.txt
examples
java
libraries
tools-builder
arduino.l4j.ini
arduino_debug.l4j.ini
libusb0.dll
msvcr100.dll
wrapper-manifest.xml

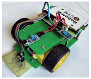
- **Install AvrStudio** – double-click on `AvrStudio4.19.730Setup.exe` icon, agree on the license agreement and use all the default settings.

5. Software Development Steps



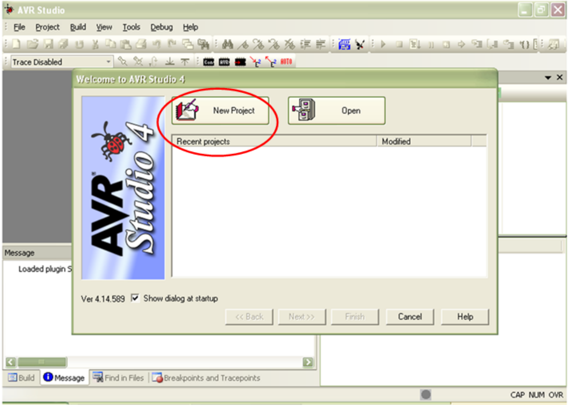
- Steps in producing executable machine code & loading to RoboKar brain:
  1. Create project using AvrStudio.
  2. Configure Compiler, the HAL & the MicroC/OS-II object files.
  3. Enter, edit & save source files.
  4. Compile/build project with GNU C
  5. Upload machine code to RoboKar

5. Software Development Steps

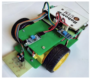


① Create project using AvrStudio.

- Run AVR Studio & select **New Project**.

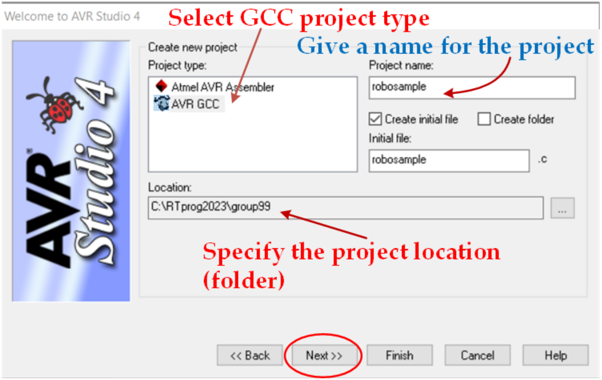


5. Software Development Steps



① Create project using AvrStudio.

- Select project type, name & location:



Welcome to AVR Studio 4

Select GCC project type

Give a name for the project

Create new project

Project type:
- Atmel AVR Assembler
- AVR GCC

Project name: robosample

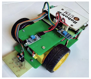
☑ Create initial file ☐ Create folder

Initial file: robosample .c

Location: C:\RTprog2023\group99

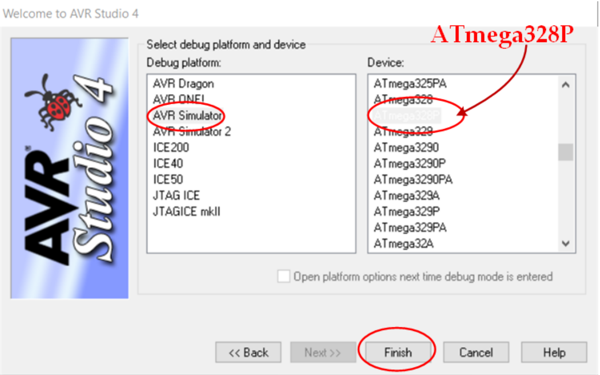
Specify the project location (folder)

5. Software Development Steps



① Create project using AvrStudio.

- Choose microcontroller (device) type: **ATmega328P**



Welcome to AVR Studio 4

Select debug platform and device

Debug platform:

- AVR Dragon
- AVR ONE!
- AVR Simulator
- AVR Simulator 2
- ICE200
- ICE40
- ICE50
- JTAG ICE
- JTAGICE mkII

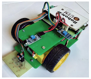
Device:

- ATmega325PA
- ATmega328
- ATmega328P
- ATmega329
- ATmega3290
- ATmega3290P
- ATmega3290PA
- ATmega329A
- ATmega329P
- ATmega329PA
- ATmega32A

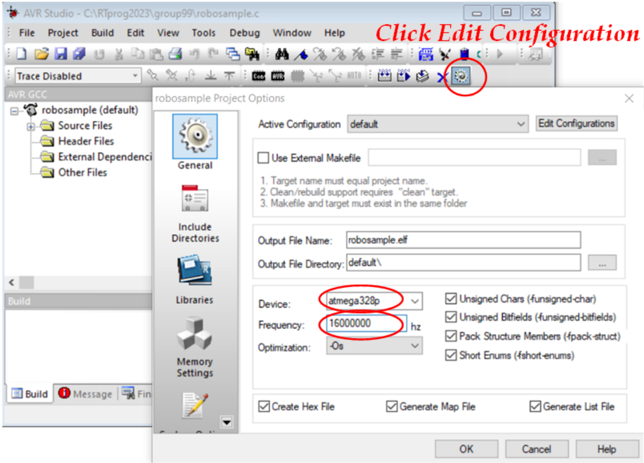
ATmega328P

□ Open platform options next time debug mode is entered

5. Software Development Steps



② Configure Compiler, the HAL & the MicroC/OS-II object files.



Click Edit Configuration

AVR Studio - C:\RTprog2023\group99\robosample.c

robosample Project Options

Active Configuration: default

☐ Use External Makefile

1. Target name must equal project name.
2. Clean/rebuild support requires "clean" target.
3. Makefile and target must exist in the same folder

Output File Name: robosample.elf

Output File Directory: default\

Device: atmega328p

Frequency: 16000000 hz

Optimization: -Os

☑ Unsigned Chars (-funsigned-char)

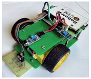
☑ Unsigned Bitfields (-funsigned-bitfields)

☑ Pack Structure Members (-fpack-struct)

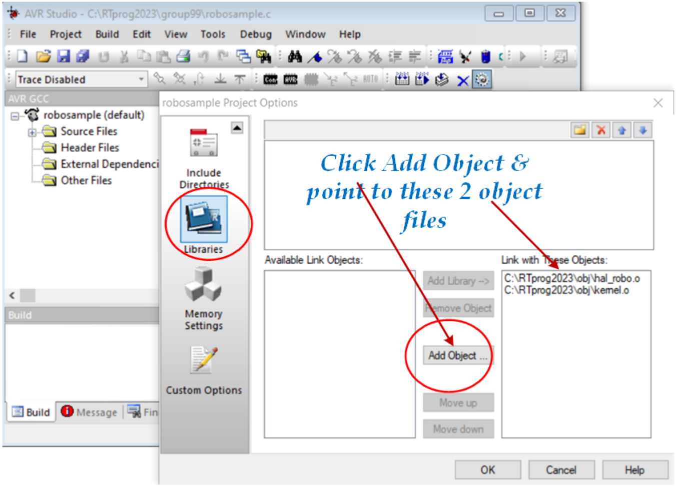
☑ Short Enums (-fshort-enums)

☑ Create Hex File ☑ Generate Map File ☑ Generate List File

5. Software Development Steps



② Configure Compiler, the HAL & the MicroC/OS-II object files.



AVR Studio - C:\RTprog2023\group99\robosample.c

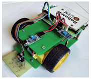
robosample Project Options

Click Add Object & point to these 2 object files

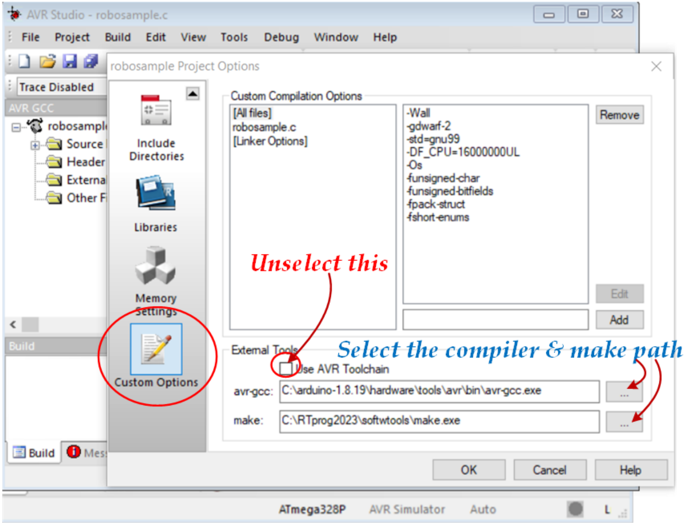
Link with These Objects:

C:\RTprog2023\obj\hal_robo.o
C:\RTprog2023\obj\kernel.o

5. Software Development Steps



② Configure Compiler, the HAL & the MicroC/OS-II object files.



Custom Compilation Options

[All files]
robosample.c
[Linker Options]

-Wall
-gdwarf-2
-std=gnu99
-DF_CPU=16000000UL
-Os
-funsigned-char
-funsigned-bitfields
-fpack-struct
-fshort-enums

External Tools

□ Use AVR Toolchain

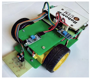
avr-gcc: C:\arduino-1.8.19\hardware\tools\avr\bin\avr-gcc.exe

make: C:\RTprog2023\softwtools\make.exe

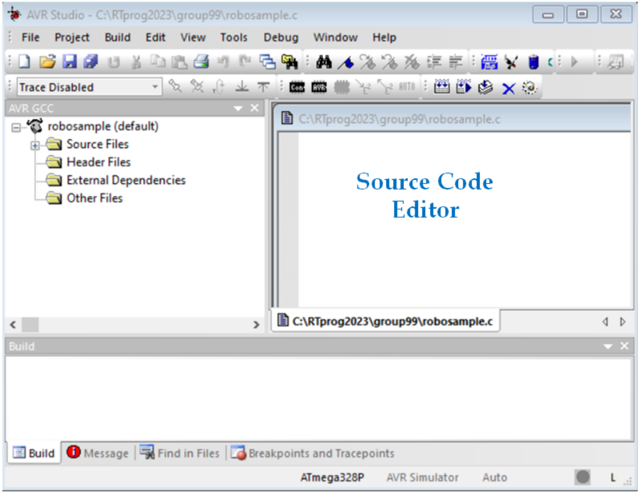
Unselect this

Select the compiler & make path

5. Software Development Steps



③ Enter, edit & save source files in AvrStudio.



AVR Studio - C:\RTprog2023\group99\robosample.c

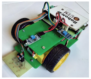
C:\RTprog2023\group99\robosample.c

Source Code Editor

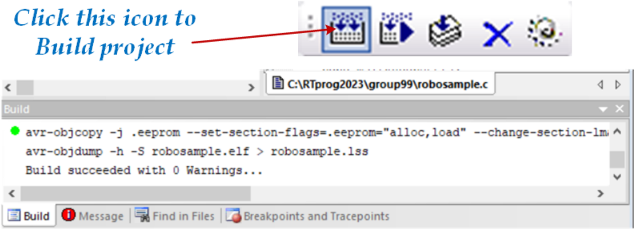
C:\RTprog2023\group99\robosample.c
Build
Build
Message
Find in Files
Breakpoints and Tracepoints
ATmega328P
AVR Simulator
Auto

5. Software Development Steps

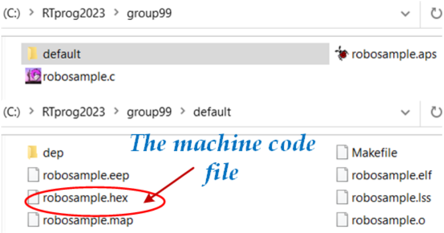
④ Compile/build project with GNU C



Click this icon to Build project

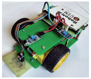

C:\RTprog2023\group99\robosample.c
Build
Build
Message
Find in Files
Breakpoints and Tracepoints

```
avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" --change-section-lma
avr-objdump -h -S robosample.elf > robosample.lss
Build succeeded with 0 Warnings...
```



The machine code file

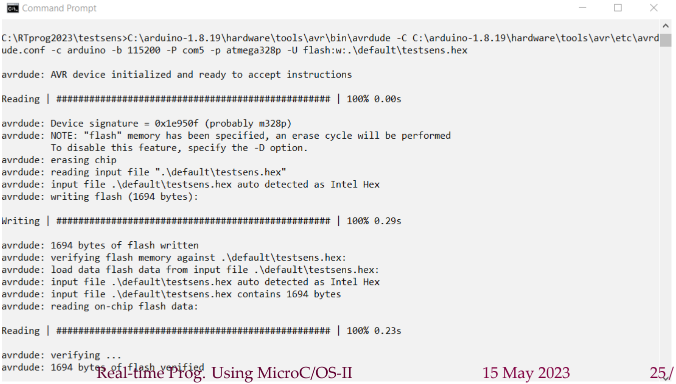
5. Software Development Steps



⑤ Upload machine code to RoboKar

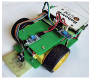
- Connect the USB cable to RoboKar, open a command prompt in the project folder, use `Avrdude.exe` (included with Arduino IDE) or use the `prog.bat` batch file provided to automate the upload process. Change the **COM** port & the **machine code name** according to your project.

```
C:\RTprog2023\group99>C:\arduino-1.8.19\hardware\tools\avr\bin\avrdude -C C:\arduino-1.8.19\
hardware\tools\avr\etc\avrdude.conf -c arduino -b 115200 -P com5 -p atmega328p -U flash:w:.\
default\robosample.hex
```



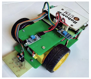
```
C:\RTprog2023\testsens>C:\arduino-1.8.19\hardware\tools\avr\bin\avrdude -C C:\arduino-1.8.19\hardware\tools\avr\etc\avrd
ude.conf -c arduino -b 115200 -P com5 -p atmega328p -U flash:w:.\default\testsens.hex

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: NOTE: "flash" memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file ".\default\testsens.hex"
avrdude: input file .\default\testsens.hex auto detected as Intel Hex
avrdude: writing flash (1694 bytes):

Writing | ################################################## | 100% 0.29s

avrdude: 1694 bytes of flash written
avrdude: verifying flash memory against .\default\testsens.hex:
avrdude: load data flash data from input file .\default\testsens.hex:
avrdude: input file .\default\testsens.hex auto detected as Intel Hex
avrdude: input file .\default\testsens.hex contains 1694 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.23s

avrdude: verifying ...
avrdude: 1694 bytes of flash verified
```

6. RoboKar Code Template



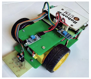
```
/*
 *   ROBOSAMPLE.C -- A sample/template for RoboKar program with uCOS-II
 *   Written by: Rosbi Mamat 6/5/2014
 *   Updated : 1/5/2023 Modified to show proximity & light sensor usage
 */

#include "..\inc\kernel.h"                  /* Always include these to use uCOS-II       */
#include "..\inc\hal_robo.h"                /*   and RoboKar HAL                         */

#define TASK_STK_SZ            128          /* Size of each task's stacks (# of bytes)   */
#define TASK_START_PRIO          1          /* Highest priority                          */
#define TASK_CHKCOLLIDE_PRIO     2
#define TASK_CTRLMOTOR_PRIO      3
#define TASK_NAVIG_PRIO          4          /* Lowest priority                           */

OS_STK TaskStartStk[TASK_STK_SZ];           /* TaskStartTask stack                       */
OS_STK ChkCollideStk[TASK_STK_SZ];          /* Task StopOnCollide stack                  */
OS_STK CtrlmotorStk[TASK_STK_SZ];           /* Task CtrlMotors stack                     */
OS_STK NavigStk[TASK_STK_SZ];               /* Task NavigRobot stack                     */

/* ------ Global shared variable -------*/
/* Ideally, this should be protected by a semaphore etc */
struct robostate
{
    int rspeed;                             /* right motor speed  (-100 -- +100)         */
    int lspeed;                             /* leftt motor speed  (-100 -- +100)         */
    char obstacle;                          /* obstacle? 1 = yes, 0 = no                 */
} myrobot;
```

6. RoboKar Code Template



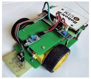
```
/*------High pririority task----------*/
void CheckCollision (void *data)
{
    for(;;)
    {
        if ( (robo_proxSensor() == 1) )             /* obstacle?                          */
            myrobot.obstacle = 1;                   /* signal obstacle present            */
        else
            myrobot.obstacle = 0;                   /* signal no obstacle                 */

        OSTimeDlyHMSM(0, 0, 0, 100);                /* Task period ~ 100 ms               */
    }
}

/* Control robot Motors TASK */
void CntrlMotors (void *data)
{
    int speed_r, speed_l;

    for(;;)
    {
        speed_r = myrobot.rspeed;
        speed_l = myrobot.lspeed;
        robo_motorSpeed(speed_l, speed_r);
        OSTimeDlyHMSM(0, 0, 0, 250);                /* Task period ~ 250 ms               */
    }
}
```

6. RoboKar Code Template



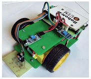
```c
/* --- Task for navigating robot ----
 * Write you own navigation task here
 */

void Navig (void *data)
{
    for (;;)
    {
        if (myrobot.obstacle == 1)                  /* If blocked then reverse              */
        {
            myrobot.rspeed   = -LOW_SPEED;          /* REVERSE */
            myrobot.lspeed   = -LOW_SPEED;
        }
        else                                        /* obstacle is far away & no collision   */
        {
            myrobot.rspeed   = MEDIUM_SPEED;        /* move forward with medium speed        */
            myrobot.lspeed   = MEDIUM_SPEED;
        }

        if (robo_lightSensor() > 60)                /* it is too bright, I'm photophobia     */
        {
            myrobot.rspeed   = -LOW_SPEED;          /* turn right to avoid                   */
            myrobot.lspeed   =  LOW_SPEED;
        }
        OSTimeDlyHMSM(0, 0, 0, 500);                /* Task period ~ 500 ms                  */
    }
}
```

6. RoboKar Code Template



```
/*------Highest priority task----------*/
/* Create all other tasks here          */
void TaskStart( void *data )
{
    OS_ticks_init();                                        /* enable RTOS timer tick        */

    OSTaskCreate(CheckCollision,                            /* Task function                 */
                (void *)0,                                  /* nothing passed to task        */
                (void *)&ChkCollideStk[TASK_STK_SZ - 1],    /* stack allocated to task       */
                TASK_CHKCOLLIDE_PRIO);                      /* priority of task              */

    OSTaskCreate(CntrlMotors,                               /* Task function                 */
                (void *)0,                                  /* nothing passed to task        */
                (void *)&CtrlmotorStk[TASK_STK_SZ - 1],     /* stack allocated to task       */
                TASK_CTRLMOTOR_PRIO);                       /* priority of task              */

    OSTaskCreate(Navig,                                     /* Task function                 */
                (void *)0,                                  /* nothing passed to task        */
                (void *)&NavigStk[TASK_STK_SZ - 1],         /* stack allocated to task       */
                TASK_NAVIG_PRIO);                           /* priority of task              */

    while(1)
    {
        OSTimeDlyHMSM(0, 0, 5, 0);                          /* Task period ~ 5 secs          */
        robo_LED_toggle();                                  /* Show that we are alive        */
    }

}
```

6. RoboKar Code Template



```
int main( void )
{
    robo_Setup();                                          /* initialize HAL for RoboKar     */
    OSInit();                                              /* initialize UCOS-II kernel      */

    robo_motorSpeed(STOP_SPEED, STOP_SPEED);               /* Stop the robot                 */
    myrobot.rspeed   = STOP_SPEED;                         /* Initialize myrobot states      */
    myrobot.lspeed   = STOP_SPEED;
    myrobot.obstacle = 0;                                  /*  No collisioin                 */

    OSTaskCreate(TaskStart,                                /* create TaskStart Task          */
                (void *)0,
                (void *)&TaskStartStk[TASK_STK_SZ - 1],
                TASK_START_PRIO);
    robo_Honk(); robo_wait4goPress();                      /* Wait for to GO                 */
    OSStart();                                             /* Start multitasking             */
    while (1);                                             /* die here                       */
}
```