Project

Goal: Implementing a basic control system for an autonomous ground vehicle navigating in a simple racetrack.

Firmware requirements

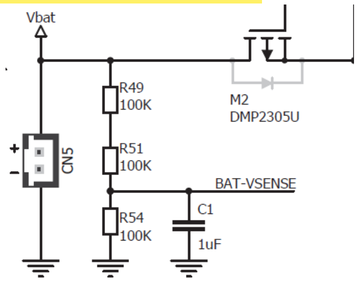
- Main loop requirements
  - The control loop should be implemented at 1 kHz frequency.
  - The motor PWM should be updated at 1 kHz and the IR sensor should be read at 1 kHz frequency.
  - Initially, the robot should be in “Wait for start” state.
    - In this state, the PWM DC of all the motors should be 0.
    - The LED A0 and should blink at 1 Hz frequency.
  - Once the button RE8 is pressed, the robot should go in the “Moving” state.
    - In this state, the PWM is generated to navigate a racetrack.
    - Two commanded percentages should be computed:
      - *surge* [0 to 100%], with 100% meaning full speed forward, and 0% meaning no forward motion.
      - *yaw_rate* [0 to 100%], with 100% meaning a clockwise rotation and 0% no rotation.
    - The algorithm to compute them should be:
      - If the sensed distance is below a threshold MINTH, the robot should turn **clockwise** on the spot.
      - If the sensed distance is above a MAXTH threshold, the robot should go forward.
      - Default values for MINTH and MAXTH should be implemented.
      - In between the MINTH and MAXTH, groups are free to choose a proportional law, combining forward motion and turning as a function of the distance, or a hysteresis mechanism (rotation on the spot until the MAXTH is reached, then going forward until below the MINTH).
    - The LED A0 should blink at 1 Hz frequency.
  - If the button RE8 is pressed for a second time, the robot should go back in the “Wait for start” state.
- Motor control
  - Once the *surge* and *yaw_rate* signals have been generated, the allocation to the four wheels must be considered. To do so, the *left_pwm* and *right_pwm* signals must be computed taking the *surge* signal and summing or subtracting the *yaw_rate* signal depending on the left/right side. If the resulting *left_pwm* or *right_pwm* signals are greater than +-100%, both values should be scaled down by the same factor to stay in the range [-100% to 100%].
  - Four PWM signals must be generated to control the buggy, on pins RD1 to RD4 (PR65 to RP68) using four Output Compare peripherals.
  - The frequency of the PWM signals must be 10 kHz.
  - The actuation of the wheels follows the specification reported in the table below:

Command	**PWM with DC > 0**	**PWM with DC = 0**
Left wheels forward (left_pwm > 0)	RD2 = left_pwm	RD1 = 0
Left wheels backward (left_pwm < 0)	RD1 = -left_pwm	RD2 = 0
Right wheels forward (right_pwm > 0)	RD4 = right_pwm	RD3 = 0
Right wheels backward (right_pwm < 0)	RD3 = -right_pwm	RD4 = 0

- Lights
    - The buggy lights should be controlled depending on the state according to the following tables:

State “Wait for start”	Left	Right	Brakes	Low intensity	Beam lights
All the time	Blink 1 Hz	Blink 1 Hz	Off	Off	Off
State “Moving”	**Left**	**Right**	**Brakes**	**Low intensity**	**Beam lights**
Forward (surge > 50%)	-	-	Off	Off	On
Slow or stationary (Surge < 50%)	-	-	On	On	Off
Turning clockwise (yaw_rate > 15%)	Off	Blink 1 Hz	-	-	-
No turning (yaw_rate < 15%)	Off	Off	-	-	-

- Battery sensing
    - The voltage of the battery (BAT-VSENSE in the figure below) is available on pin AN11. It is sensed after a partitioning circuit, i.e., in between a 200 kohm resistor and a 100 kohm resistor.



- IR sensor
    - The infrared sensor should be mounted on the *Buggy Mikrobus* 2 (i.e., in front of the buggy). The signal can be read on AN14, while the enable to the IR sensor must be given on the digital I/O on RB9.
- Data logging / command interface through UART
    - The UART to RS232 module should be installed on the *Clicker Mikrobus* 2. The TX signal should be remapped to RD0/RP64, while the RX signal should be remapped to RD11/RPI75.
    - The microcontroller should send, to the PC, the following messages (in all the states)
        - $MBATT,v_batt* where v_batt is the sensed battery in Volt, at 1 Hz frequency. Use two digits, i.e., X.YZ
        - $MDIST,distance* where distance is the sensed distance in cm, at 10 Hz frequency. Use an integer.
        - $MPWM,dc1,dc2,dc3,dc4* where dc1 is the duty cycle on RD1, dc2 is the duty cycle on RD2, etc. at 10 Hz frequency. Use integers.
    - The microcontroller should receive, from the PC, the following messages (in all the states)
        - $PCTH,minth,maxth*, where minth is the MINTH threshold (in cm), and maxth is the MAXTH (in cm) threshold to be set. Both values will be integers.
    - Given the chosen UART baud rate, the firmware should never lose a message due to its implementation (i.e., proper dimensioning of buffers), even with full use of the bandwidth.

Evaluation criteria

Among other things, these criteria will be used:

- Adherence to the provided specifications
- Correctness of the interrupts service routines
- Correct handling of shared data
- Management of the UART FIFO and circular buffers on both sending and receiving
- General code cleanliness

Pin Mapping

- RB8 Left side lights
- RF1 Right-side lights
- RF0 brakes
- RG1 low intensity lights
- RA7 beam headlights
- AN11 battery sensing
- RD1/RP65 left PWM backward motion
- RD2/RP66 left PWM forward motion
- RD3/RP67 right PWM backward motion
- RD4/RP68 right PWM forward motion
- AN14 IR sensor voltage
- RB9 IR sensor enable
- RD0/RP64 UART TX

- RD11/RPI75 UART RX

Hardware setup