Design with MicroProcessors

-project-

XMR – eXtra Medium Robot

Documentation

1. **Specification**

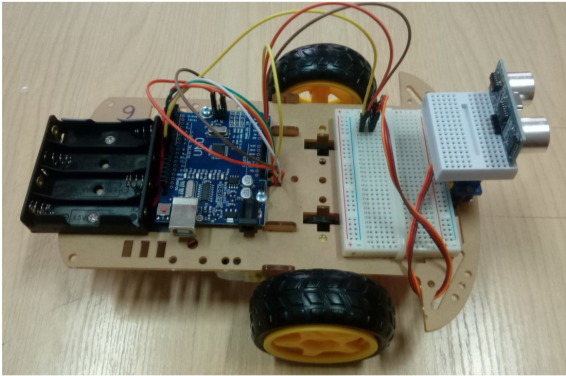
Our project involves configuring the given robot both from a hardware and a software perspective to make it perform some tasks. The tasks our robot can perform are: hardware test, Bluetooth control mode, obstacle avoid, object follow.

* Hardware test: the robot moves forward while also moving its servo motor and reading data from the ultrasonic sensor.
* Bluetooth control mode: the robot can be controlled from a mobile device through its Bluetooth module. The mobile device uses the “Bluetooth RC Controller” app (available on Google Play Store). The robot can also emit sounds.
* Obstacle avoid: the robot moves and tries to avoid each obstacle it detects. When an obstacle is detected, the robot tries to go left or right and if both paths are blocked, turns around.
* Object follow: the robot follows an object. If the object cannot be detected anymore, the robot looks continuously for it (left and right) until it is found again and it can be followed.

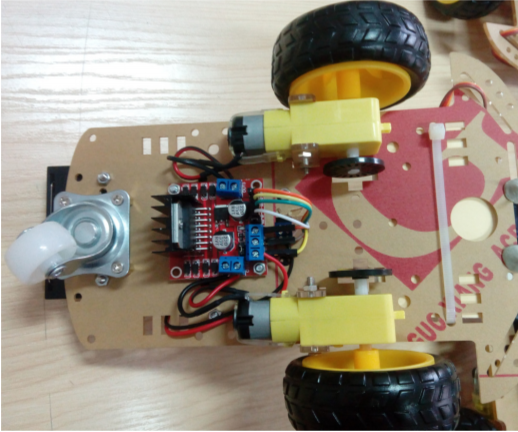
1. **Hardware modifications**

In order to accomplish the desired goals, the robot needed some hardware additions. These are:

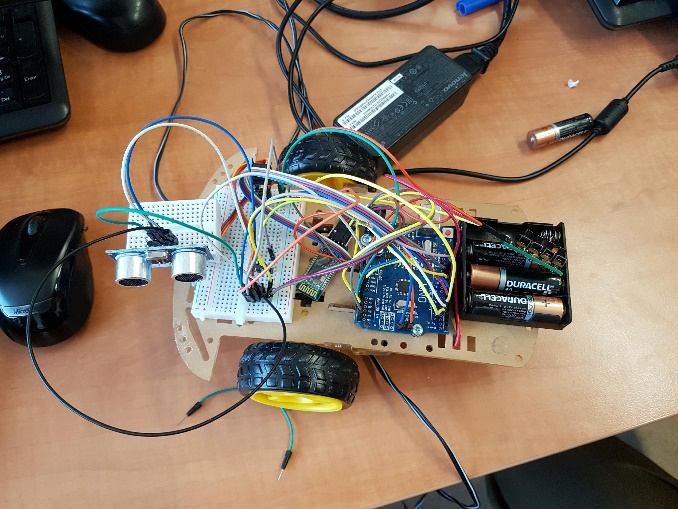
* Connected the ultrasonic sensor to the Arduino Uno board.
* Added switches (to select between the operation modes).
* Added a Bluetooth module.
* Added a buzzer.



Before the modifications – top view



Before the modifications – bottom view



After the modifications – top view

1. **Implementation outlines**

The implementation consists of a number of .cpp files together with associated headers for each functionality, also having some headers for constants and the .ino file created by the Arduino IDE.

* xmr.ino is the file where the “main” code is written (the setup() and loop() functions). The setup() is used to do the initializations needed by each mode (by calling their setup functions). The loop() first calls the getMode() function to get the mode of operation, the calls the appropriate loop function. One of the loop functions, loopTest(), is also present in this file. It is used to test the basic functionalities of the motors and the ultrasonic sensor.
* avoid.cpp and avoid.h: These files describe the avoid and follow functionalities (which are implemented together since they are “mirrored” – follow is mostly avoid which reversed decisions). Its “exposed” functions are loopAvoid(int) and setupAvoid(). The loopAvoid(int) function receives an int as parameter to specify the behaviour: 1- follow, 0 - avoid.

//returns the decision (turn, go forward, do nothing etc) based on the employed strategy and sensor readings

int getDecision(int strategy){

int decision = NOTHING;

if(shouldGo(lookFront, strategy)){

stop();

if(shouldGo(lookLeft, strategy)){

if(shouldGo(lookRight, strategy)){

if (strategy == AVOID){

decision = TURN;

}

else{

decision = NOTHING;

}

}

else{

decision = GO\_RIGHT;

}

}

else{

decision = GO\_LEFT;

}

}

else{

decision = GO\_FRONT;

}

return (decision);

}

void loopAvoid(int strategy){

decision = getDecision(strategy); // get the decision

switch (decision){

case GO\_RIGHT:

turn(avoidSpeed,turnDelay);

break;

case GO\_LEFT:

turn(-avoidSpeed,turnDelay);

break;

case TURN:

turn(avoidSpeed,turnDelay\*2);

break;

case NOTHING:

stop();

break;

}

if (strategy == FOLLOW && decision == NOTHING){ //if following and nothing is seen it should wait

stop();

}

else{

drive(avoidSpeed);

};

}

* bluetooth.cpp and bluetooth.h: These files describe the interaction between the robot and a Bluetooth device (specifically one that uses the aforementioned app). It has the setupBLT() and loopBLT() functions used to setup this mode and to have it run if the mode is the active one
* modeSelector.cpp and modeSelector.h: These files describe the functionality of the switches that are used to select the operation mode. In order to do this, a new enum data type is defined (mode\_e – mode\_t) which has as possible values all the modes of operation of the robot: bluetooth, hardwareTest, obstacleAvoid, obstacleFollow. The two functions are setupModeSelector() and getMode(). The first one is used during the setup phase, while the second function is called in the loop to get the active mode and returns a mode\_t value.
* motors.cpp and motors.h: These files are used to specify the functionality of the robot’s 2 motors, implementing a variety of functions for this. As usual, setupMotors() is used in the setup phase to setup the motors. The go(int speedLeft, int speedRight) function is used to start each motor with the specified speed. This function is used as the basis for the other functions. Other functions are: drive(int speed) – causes both motors to start with the same speed (which can be negative for backward movement); turn(int speed) – causes one of the motors to start with the specified speed, while the other starts with the same speed but in the reverse direction, thus causing the robot to turn. The direction can be specified by giving as argument a positive or negative number (positive for right turn, negative for left turn). There is another similar function: turn(int speed, int delayed) which only turns the robot for the specified amount of time (for more precise turns). The final function is stop() and it simply stops both motors.

//applies speeds to motors

void go(int speedLeft, int speedRight){

if (speedLeft > 0) {

analogWrite(leftMotorPinA, speedLeft);

analogWrite(leftMotorPinB, 0);

}

else {

analogWrite(leftMotorPinA, 0);

analogWrite(leftMotorPinB, -speedLeft);

}

if (speedRight > 0) {

analogWrite(rightMotorPinA, speedRight);

analogWrite(rightMotorPinB, 0);

}else {

analogWrite(rightMotorPinA, 0);

analogWrite(rightMotorPinB, -speedRight);

}

}

* pinConfig.h: this header is used to specify the name of each pin used in the project.
* sysConfig.h: this header is used to specify all the constants used in the project.