Лабораторна робота №1

Тема: Програмування робота Pololu m3pi.

Мета: Розібратися з будовою та навчитися програмувати робота Pololu m3pi.

Завдання: Змінити програму зменшивши або збільшивши швидкість руху

BEAM роботи - BEAM є абревіатурою від Biology, Electronics, Aesthetics, Mechanics. Це термін, що позначає принцип побудови роботів, що використовує прості аналогові ланцюги (наприклад, компаратори) замість мікропроцесорів з метою досягти незвично простого (в порівнянні з традиційними пересувним роботами) дизайну, Який жертвує гнучкістю ради надійності і ефективності виконання виразно завдання. Однак, існують винятки, які використовують Не тільки аналогові ланцюги (звані «мутантами»). Роботи BEAM зазвичай представляють собою набір вищезгаданих аналогових ланцюгів (копіює біологічні нейрони), які дозволяють роботу взаємодіяти з робочим оточенням.

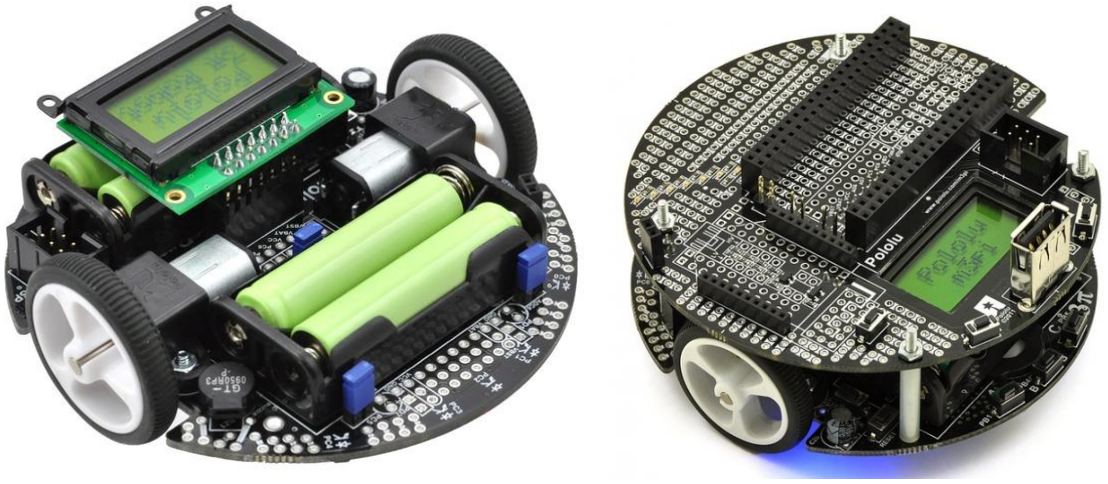


Рис. 1 - Робот Pololu m3pi, розширена версія

Набір для створення 3pi від американської компанії Pololu. Це повноцінна, високопродуктивна мобільна платформа з двома мікромоторами з редуктором, п'ятьма датчиками відображення, LCD екраном 2 рядки по 8 символів, динаміком і трьома кнопками для різних потреб (програмуються користувачем). Все це підключено до мікроконтролеру ATmega328, який можна програмувати на зручному вам мовою (C, C ++ і звичайно ж Arduino). Робот здатний розвивати швидкість до метра в секунду. Добре підходить для початківців роботостроітелей і для тих, хто вже награвся звичайними непрограмовані роботами.

Спочатку робот проектувався для участі в змаганнях за програмами "слідуй за лінією" або "знайди вихід з лабіринту", що не заважає зробити з нього що-небудь ще, наприклад домашнього охоронця (просто примотати до нього ізолентою плазмову гармату). Робот досить компактний (9.5 см в діаметрі, 83 гр. Ваги без батарей), працює від 4 батарейок розміру AAA (не входять в поставку), в той час як унікальна система живлення забезпечує на моторах постійні 9.25В

незалежно від ступеня розрядки батарей, дозволяючи розвинути швидкість до 1 метра в секунду, забезпечуючи точні повороти і розвороти незалежно від напруги батарей (само-собою до тих пір, поки вони не сядуть зовсім).

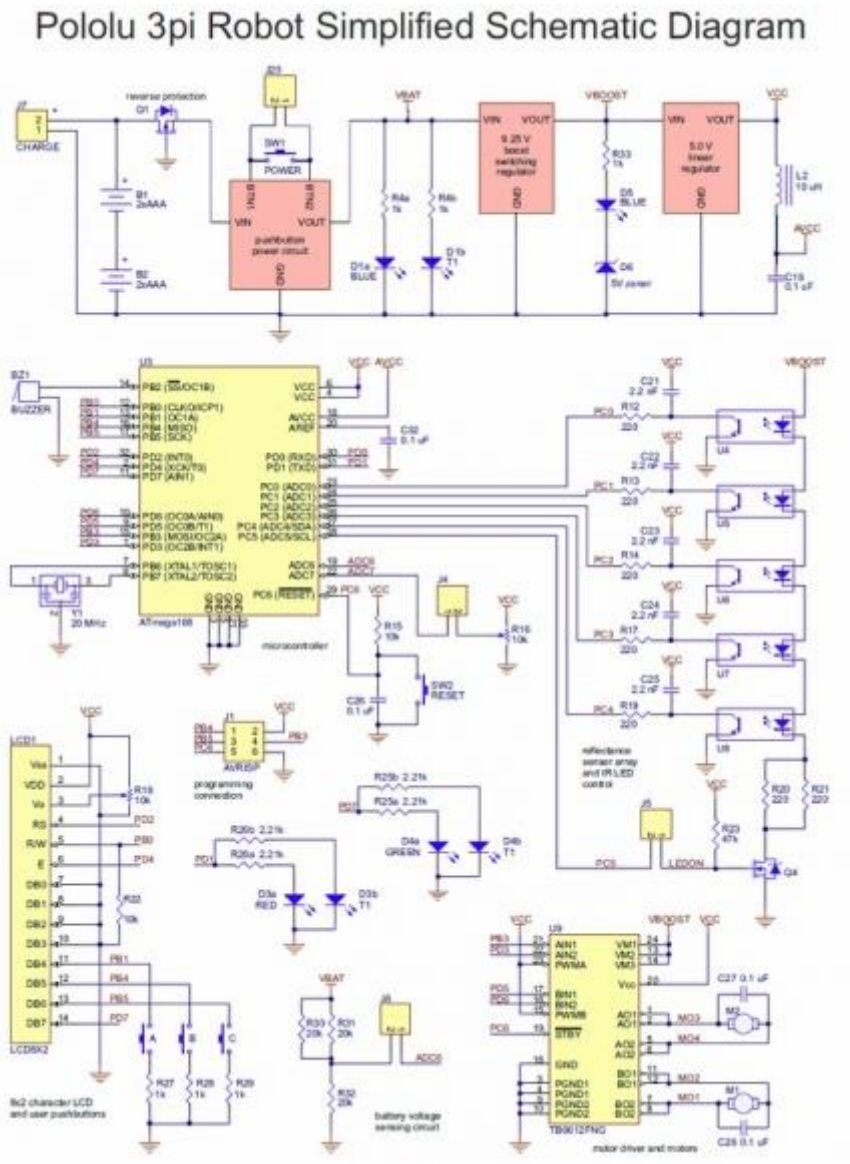


Рис. 2 - Принципово-електрична схема Pololu m3pi

Програмується робот на мові C / C ++ з AVR Studio (можна і на Асемблері, якщо ви справжній джедай) за допомогою вибору програм. Так само можна писати прошивку на мові Arduino і зашивати її з Arduino IDE, але для цього знову таки знадобиться програматор. Мозком робота виступає мікроконтролер Atmel ATmega328P (такий же як в Arduino Uno, Nano і ін.), Що працює на частоті 20 МГц (тобто трохи спритніше вищезазначених Ардуіно працюють на 16 МГц) і несе на борту 32 кб пам'яті під програми, 2 кб ОЗУ і 1 кб незалежній пам'яті EEPROM. Популярні C / C ++ компілятори відмінно працюють з 3pi, Atmel Studio надає комфортабельну середовище розробки і широкий набір бібліотек від Pololu надають інтерфейс для доступу до всього інтегрованого в роботі залозу. Як вже говорилося, писати програми для 3pi можна і за допомогою популярної середовища розробки Arduino IDE, виробник надає велику кількість готових прикладів як для знайомства з різноманітними компонентами 3pi, так і для більш

комплексного поведінки, такого як слідування за лінією і пошук виходу з лабіринту.

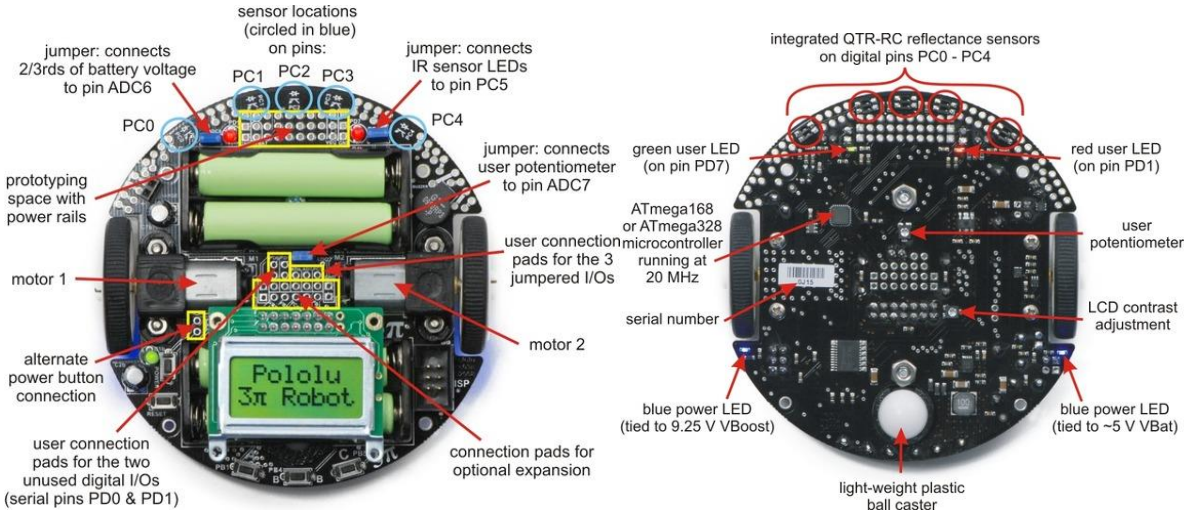
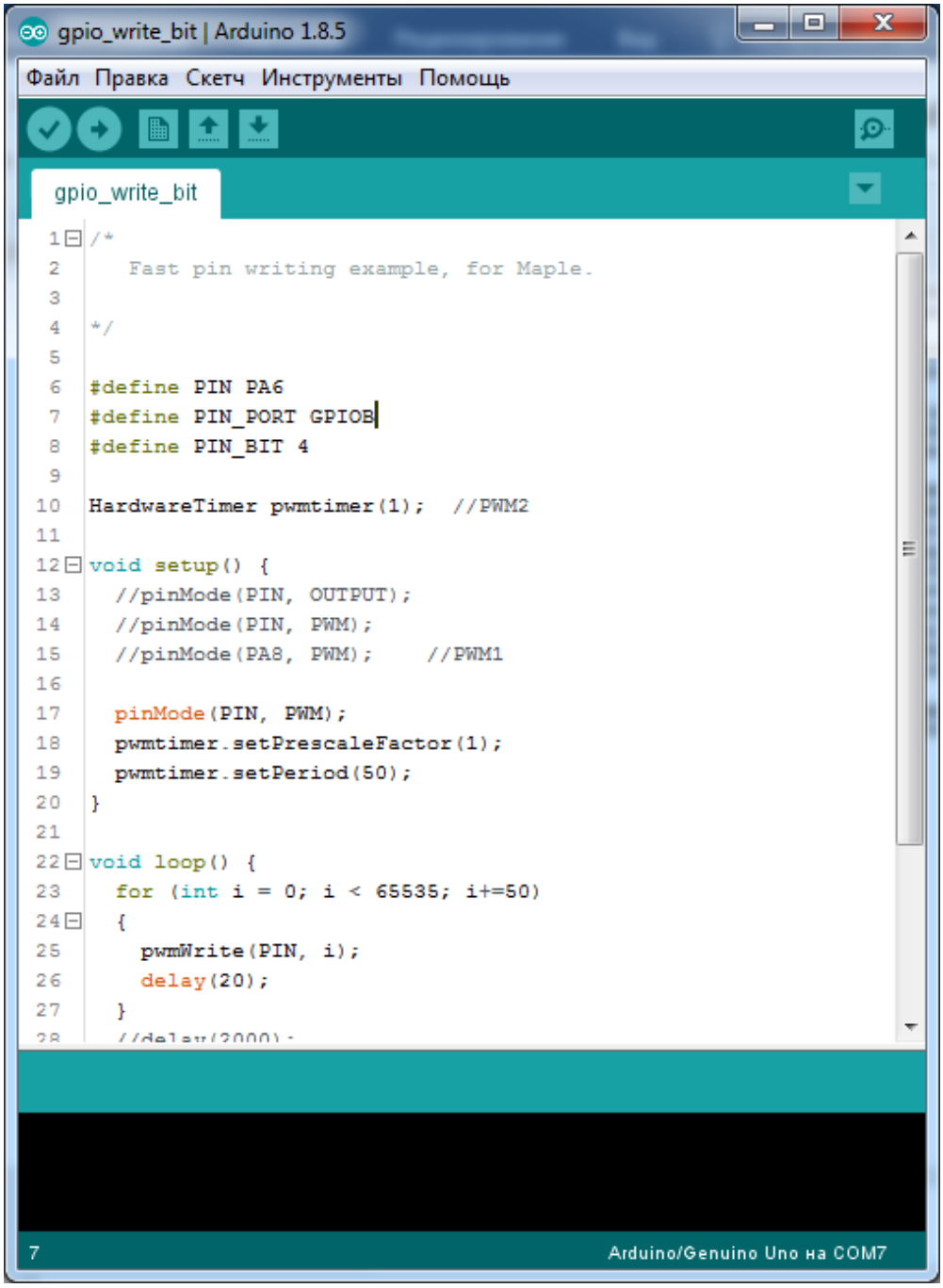


Рис. 3 – Розміщення елементів на платі

Інтерфейс програми Arduino для програмування контролера

Arduino (Ардуіно) — апаратна обчислювальна платформа для аматорського конструювання, основними компонентами якої є плата мікроконтролера з елементами вводу/виводу та середовище розробки Processing/Wiring на мові програмування, що є підмножиною C/C++.Arduino може використовуватися як для створення автономних інтерактивних об'єктів, так і підключатися до програмного забезпечення, яке виконується на комп'ютері (наприклад: Processing, Adobe Flash, Max/MSP, Pure Data, SuperCollider).



Інформація про плату (рисунок друкованої плати, специфікації елементів, програмне забезпечення) знаходяться у відкритому доступі і можуть бути використані тими, хто воліє створювати плати власноруч.

Приклад коду для руху і слідкуванню за лінією

```
// The following libraries will be needed by this demo
#include <Pololu3pi.h>
#include <PololuQTRSensors.h>
#include <OrangutanMotors.h>
#include <OrangutanAnalog.h>
#include <OrangutanLEDs.h>
#include <OrangutanLCD.h>
#include <OrangutanPushbuttons.h>
#include <OrangutanBuzzer.h>

Pololu3pi robot;
unsigned int sensors[5]; // an array to hold sensor values
unsigned int last_proportional = 0;
long integral = 0;


// This include file allows data to be stored in program space.  The
// ATmega168 has 16k of program space compared to 1k of RAM, so large
// pieces of static data should be stored in program space.
#include <avr/pgmspace.h>

// Introductory messages.  The "PROGMEM" identifier causes the data to
// go into program space.

const char welcome_line1[] PROGMEM = " Pololu";
const char welcome_line2[] PROGMEM = "3\xf7 Robot";
const char demo_name_line1[] PROGMEM = "PID Line";
const char demo_name_line2[] PROGMEM = "follower";
```

```
// A couple of simple tunes, stored in program space.
const char welcome[] PROGMEM = ">g32>>c32";
const char go[] PROGMEM = "L16 cdegreg4";

// Data for generating the characters used in load_custom_characters
// and display_readings.  By reading levels[] starting at various
// offsets, we can generate all of the 7 extra characters needed for a
// bargraph.  This is also stored in program space.
const char levels[] PROGMEM = {
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b11111,
  0b11111,
  0b11111,
  0b11111,
  0b11111,
  0b11111,
  0b11111
};

// This function loads custom characters into the LCD.  Up to 8
// characters can be loaded; we use them for 7 levels of a bar graph.
void load_custom_characters()
{
  OrangutanLCD::loadCustomCharacter(levels + 0, 0); // no offset, e.g. one bar
```

```
  OrangutanLCD::loadCustomCharacter(levels + 1, 1); // two bars
  OrangutanLCD::loadCustomCharacter(levels + 2, 2); // etc...
  OrangutanLCD::loadCustomCharacter(levels + 3, 3);
  OrangutanLCD::loadCustomCharacter(levels + 4, 4);
  OrangutanLCD::loadCustomCharacter(levels + 5, 5);
  OrangutanLCD::loadCustomCharacter(levels + 6, 6);
  OrangutanLCD::clear(); // the LCD must be cleared for the characters to take effect
}

// This function displays the sensor readings using a bar graph.
void display_readings(const unsigned int *calibrated_values)
{
  unsigned char i;

  for (i=0;i<5;i++) {
    // Initialize the array of characters that we will use for the
    // graph.  Using the space, an extra copy of the one-bar
    // character, and character 255 (a full black box), we get 10
    // characters in the array.
    const char display_characters[10] = { ' ', 0, 0, 1, 2, 3, 4, 5, 6, 255 };

    // The variable c will have values from 0 to 9, since
    // calibrated values are in the range of 0 to 1000, and
    // 1000/101 is 9 with integer math.
    char c = display_characters[calibrated_values[i] / 101];

    // Display the bar graph character.
    OrangutanLCD::print(c);
  }
```

```
}

// Initializes the 3pi, displays a welcome message, calibrates, and
// plays the initial music.  This function is automatically called
// by the Arduino framework at the start of program execution.
void setup()
{
  unsigned int counter; // used as a simple timer

  // This must be called at the beginning of 3pi code, to set up the
  // sensors.  We use a value of 2000 for the timeout, which
  // corresponds to 2000*0.4 us = 0.8 ms on our 20 MHz processor.
  robot.init(2000);

  load_custom_characters(); // load the custom characters

  // Play welcome music and display a message
  OrangutanLCD::printFromProgramSpace(welcome_line1);
  OrangutanLCD::gotoXY(0, 1);
  OrangutanLCD::printFromProgramSpace(welcome_line2);
  OrangutanBuzzer::playFromProgramSpace(welcome);
  delay(1000);

//  OrangutanLCD::clear();
//  OrangutanLCD::printFromProgramSpace(demo_name_line1);
//  OrangutanLCD::gotoXY(0, 1);
//  OrangutanLCD::printFromProgramSpace(demo_name_line2);
//  delay(1000);

  // Display battery voltage and wait for button press
```

```
while (!OrangutanPushbuttons::isPressed(BUTTON_B))
{
  int bat = OrangutanAnalog::readBatteryMillivolts();

  OrangutanLCD::clear();
  OrangutanLCD::print(bat);
  OrangutanLCD::print("mV");
  OrangutanLCD::gotoXY(0, 1);
  OrangutanLCD::print("Press B");

  delay(100);
}

// Always wait for the button to be released so that 3pi doesn't
// start moving until your hand is away from it.
OrangutanPushbuttons::waitForRelease(BUTTON_B);
delay(1000);

// Auto-calibration: turn right and left while calibrating the
// sensors.
for (counter=0; counter<80; counter++)
{
  if (counter < 20 || counter >= 60)
    OrangutanMotors::setSpeeds(40, -40);
  else
    OrangutanMotors::setSpeeds(-40, 40);

  // This function records a set of sensor readings and keeps
  // track of the minimum and maximum values encountered.  The
  // IR_EMITTERS_ON argument means that the IR LEDs will be
```

```
  // turned on during the reading, which is usually what you
  // want.
  robot.calibrateLineSensors(IR_EMITTERS_ON);

  // Since our counter runs to 80, the total delay will be
  // 80*20 = 1600 ms.
  delay(20);
}
OrangutanMotors::setSpeeds(0, 0);

// Display calibrated values as a bar graph.
while (!OrangutanPushbuttons::isPressed(BUTTON_B))
{
  // Read the sensor values and get the position measurement.
  unsigned int position = robot.readLine(sensors, IR_EMITTERS_ON);

  // Display the position measurement, which will go from 0
  // (when the leftmost sensor is over the line) to 4000 (when
  // the rightmost sensor is over the line) on the 3pi, along
  // with a bar graph of the sensor readings.  This allows you
  // to make sure the robot is ready to go.
  OrangutanLCD::clear();
  OrangutanLCD::print(position);
  OrangutanLCD::gotoXY(0, 1);
  display_readings(sensors);

  delay(100);
}
OrangutanPushbuttons::waitForRelease(BUTTON_B);
```

```
  OrangutanLCD::clear();

  OrangutanLCD::print("Go!");

  // Play music and wait for it to finish before we start driving.
  OrangutanBuzzer::playFromProgramSpace(go);
  //while(OrangutanBuzzer::isPlaying());
}

// The main function.  This function is repeatedly called by
// the Arduino framework.
void loop()
{
  // Get the position of the line.  Note that we *must* provide
  // the "sensors" argument to read_line() here, even though we
  // are not interested in the individual sensor readings.
  unsigned int position = robot.readLine(sensors, IR_EMITTERS_ON);

  // The "proportional" term should be 0 when we are on the line.
  int proportional = (int)position - 2000;

  // Compute the derivative (change) and integral (sum) of the
  // position.
  int derivative = proportional - last_proportional;
  integral += proportional;

  // Remember the last position.
  last_proportional = proportional;

  // Compute the difference between the two motor power settings,
```

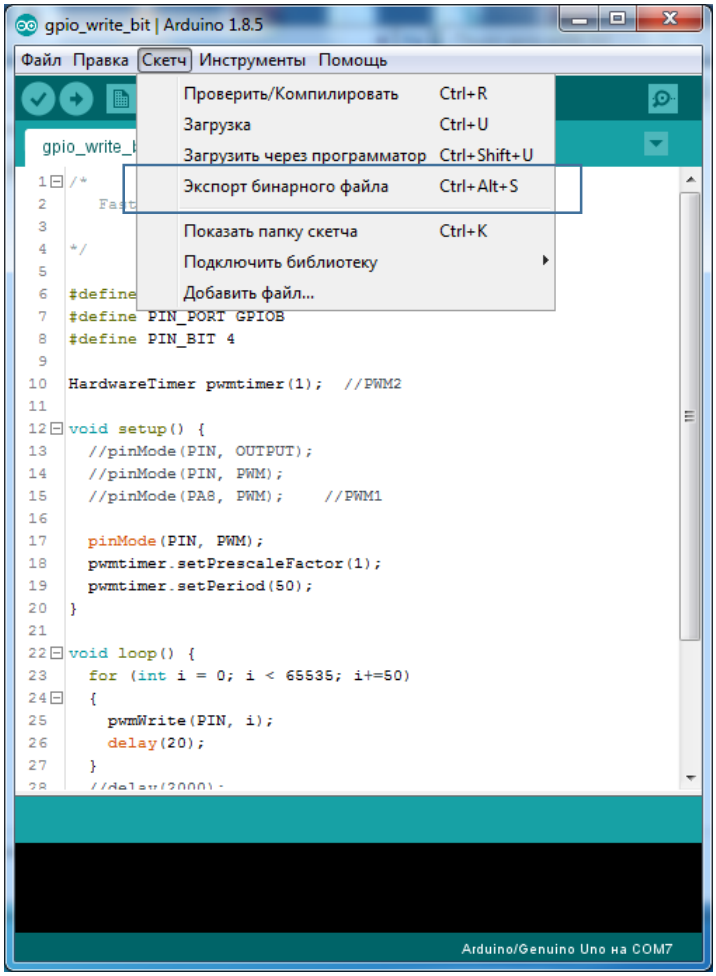
```
  // m1 - m2.  If this is a positive number the robot will turn
  // to the right.  If it is a negative number, the robot will
  // turn to the left, and the magnitude of the number determines
  // the sharpness of the turn.  You can adjust the constants by which
  // the proportional, integral, and derivative terms are multiplied to
  // improve performance.
  int power_difference = proportional/20 + integral/10000 + derivative*3/2;

  // Compute the actual motor settings.  We never set either motor
  // to a negative value.
  const int maximum = 30;
  if (power_difference > maximum)
    power_difference = maximum;
  if (power_difference < -maximum)
    power_difference = -maximum;

  if (power_difference < 0)
    OrangutanMotors::setSpeeds(maximum + power_difference, maximum);
  else
    OrangutanMotors::setSpeeds(maximum, maximum - power_difference);
}
```

Для створення файлу прошивки у форматі .hex необхідно виконати пару простих дій.

У програмі Arduino IDE переходимо в меню «Скетч» і натискаємо «Експорт бінарного файлу», або просто комбінацію клавіш Ctrl+Alt+S.



Завантаження прошивки у Pololu 3pi

Завантаження прошивки у виконується за допомогою програматора USBASP-ISP AVR V2 через програму khazama AVR Programmer.

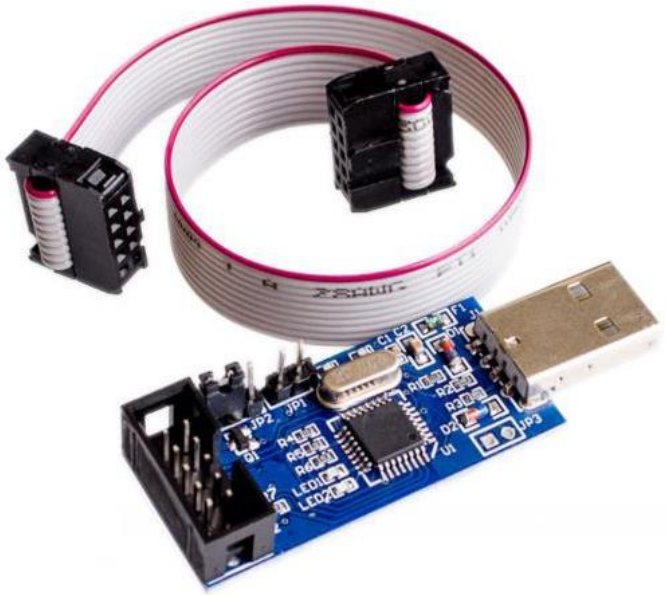
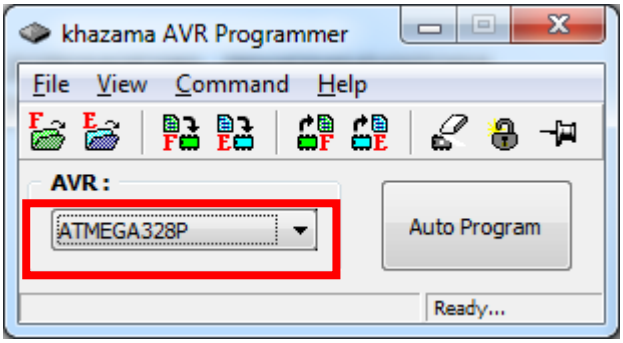
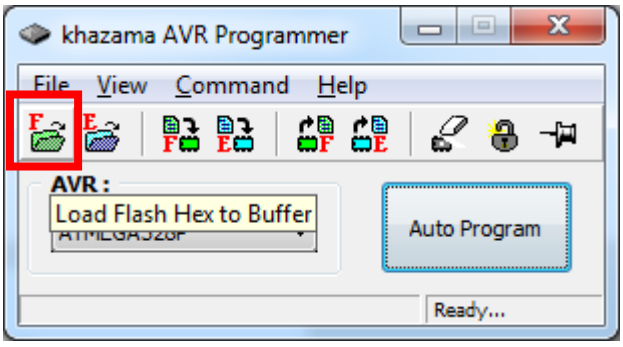


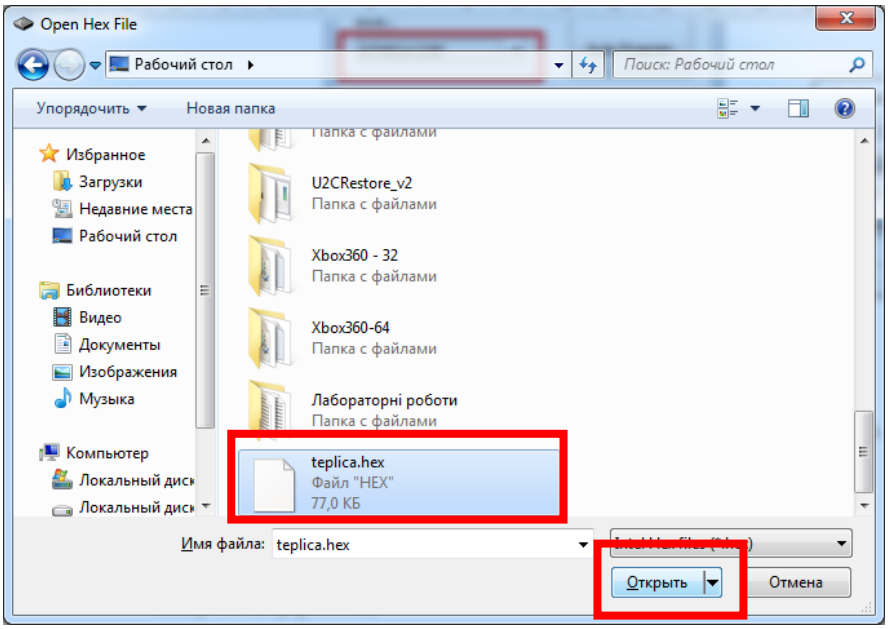
Рис. 5 - Програматор USBASP-ISP AVR V2

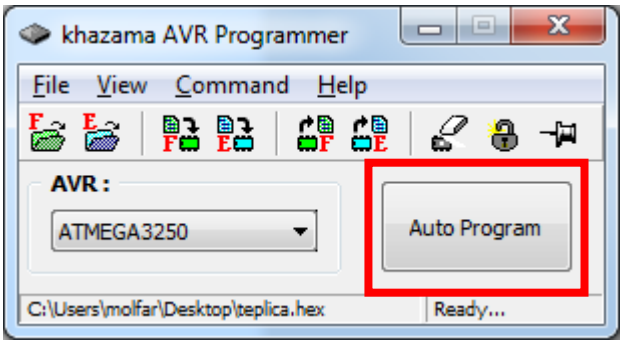


З випадаючого списку обираємо необхідний нам контролер ATmega328P.



Обираємо необхідний .hex файл прошивки та натискаємо Auto Program.





Оформлення звіту

Звіт повинен містити:

- назву та мету роботи;

- алгоритм роботи робота (стратегічний та тактичні рівні);

- опис технічних складових робота та їх характеристики (сприймаючі елементи, виконавчі механізми та ін.);

- програму завдання, написану в Arduino IDE;

- висновки по роботі.