Relazione di Progetto

"Civic Core – La Città che Comunica e Protegge"

Quando il Comitato Tecnico Scientifico ci ha proposto il tema *'Arduino e la mobilità: Progetti per la città del futuro'*, la nostra prima reazione è stata di sfida. Ci siamo chiesti: cosa significa davvero **mobilità** oggi?

Consultando il dizionario, la definizione appare semplice: *'Possibilità di spostarsi o di essere spostato'*. Tuttavia, abbiamo sentito il bisogno di evolvere questo concetto. Riflettendo abbiamo iniziato ad elaborare la nostra idea di "Città del Futuro", che non è fatta solo di metallo e cemento, che non si basa solo di spostamenti, ma di **connessioni**. Abbiamo ipotizzato quindi un ecosistema dove i servizi non sono isolati, ma "si parlano" tra loro per risolvere i problemi quotidiani della mobilità e della sicurezza. L'idea è forse frutto del fatto che nelle nostre città non sempre ci sentiamo sicuri. Da qui è nata l'idea che è centrale del nostro progetto "l'interazione": *quando l'ambiente cambia, la città risponde.*

Per cercare di dare concretezza a questa idea progettuale, abbiamo pensato di implementare tre soluzioni tecnologiche integrate. Per esigenze dovute al poco tempo disponibile ci siamo divisi in tre gruppi, ed ogni gruppo ha sviluppato una piccola soluzione e nella fase conclusiva le abbiamo interconnesse. I tre punti su cui abbiamo lavorato sono: **Stazione di monitoraggio ambientale, La Mobilità Sicura, Il Soccorso Intelligente.**

La Stazione di monitoraggio ambientale, la nostra Stazione Meteo, non è un semplice termometro. È la nostra "centrale di allarme". Il sistema legge diversi sensori e reagisce visivamente (LED e LCD) o con un allarme se i parametri escono dai limiti. Il codice raccoglie dati da quattro fonti principali: il sensore DHT11 che misura il clima dell'ambiente, il sensore MQ che rileva la presenza di gas o fumo, il sensore BH1750 per misurare l'intensità luminosa in Lux e il sensore digitale per rilevare la presenza di pioggia.

Il sistema non si limita a leggere i dati, ma prende decisioni: c'è un semaforo RGB che cambia colore in base al livello di gas rilevato**:** *Verde* per indicare Aria pulita, *Rosso per indicare* pericolo di gas sopra la soglia. Oltre a questa risposta visiva c'è anche la presenza di un Allarme tramite Buzzer che si attiva se il gas è troppo alto.

Il codice gestisce l'output in due modi: **Display LCD I2C,** abbiamo usato il protocollo I2C per il display (che ci ha permesso di risparmiare pin)**,** tale display cicla" automaticamente tra tre diverse schermate Gas, Clima, Meteo/Luce e **Monitor Seriale** che ogni 5 secondi invia un report testuale completo al computer per la diagnostica. Se i livelli di gas superano la soglia di sicurezza, il sistema avverte visivamente i cittadini (LED Rosso) e invia immediatamente un segnale acustico tramite un buzzer. È l'occhio che vigila sulla salute di chi si muove in città.

La Mobilità Sicura (Accesso RFID e Password): Abbiamo pensato a come proteggere i mezzi di trasporto dolce (bici e monopattini). L'accesso tramite **RFID** simula l'apertura di un varco pubblico o di un servizio di sharing, mentre è stata inserita anche una **serratura elettronica intelligente** con tastierino numerico e servomotore. In pratica, trasforma Arduino in una cassaforte o in un sistema di controllo accessi. La **Password** protegge una zona privata. La sicurezza è il primo passo per convincere le persone a lasciare l'auto a casa.

Il Soccorso Intelligente (Il Robot Antincendio): È il braccio operativo della città. Grazie alla tecnologia **Bluetooth**, il robot è gestibile tramite l'applicazione arduino controller bluetooth. Il robot monitora costantemente l'aria tramite il sensore collegato e rileva se il livello di gas supera la soglia, in caso affermativo scatta l'emergenza. Il robot opera in modalità "Edge Computing" (ovvero prende decisioni a bordo senza aspettare l'uomo). Con la **logica di attivazione automatica**, attiva immediatamente la **pompa d'acqua** per contrastare il pericolo. Invia un messaggio di allarme sia al computer ("!!! PERICOLO GAS !!!") sia allo smartphone via Bluetooth ("ALLARME: Gas rilevato!"). Mentre controlla il gas, il robot risponde ai comandi che vengono inviati dallo smartphone: **'F'** (Va avanti), **'B'** (Va in retromarcia), **'L'** (Gira a sinistra), **'R'** (Gira a destra), **'S'** (Si ferma completamente). È un **veicolo a guida remota con sistema di sicurezza integrato**. La pompa d'acqua

è l'unica cosa che non controlliamo: è Arduino che decide di attivarla se "sente" odore di gas, garantendo una risposta immediata.

Nello sviluppo di **Civic Core**, non tutto è stato facile. Abbiamo dovuto affrontare due problemi principali che ci hanno insegnato molto:

All'inizio, il Robot aveva difficoltà a camminare, sembrava non sopportasse il peso della cisterna d'acqua e dei vari componenti hardware. Sembrava che il peso della cisterna gravava tutto sulle ruote folli facendo slittare le ruote motrici. Abbiamo provato a spostare il baricentro (la cisterna) esattamente sopra l'asse delle ruote motorizzate per aumentare l'aderenza, senza però superare il limite di carico dei motori. Abbiamo anche pensato che poiché le ruote partono da zero con tutto il peso della cisterna abbiamo sostituito digitalWrite (che dà o 0 o 100%), con analogWrite che permette di regolare la velocità del motore.

Abbiamo imparato a usare la libreria "SoftwareSerial" per creare una corsia preferenziale per i dati. Invece di inviare testi lunghi e complicati, abbiamo creato un **linguaggio a codici**: una semplice `'A'` per l'allarme e una `'S'` per lo stato sicuro. Questo ha reso la comunicazione istantanea e a prova di errore.

La sfida delle "Soglie di Allarme" (Sensore Gas e Fumo): Il sensore del gas leggeva valori diversi a seconda dell'umidità della stanza, rischiando di far partire il Robot senza motivo.

Abbiamo scritto una funzione di **calibrazione nel codice**. Appena accendiamo la Stazione, Arduino legge il valore dell'aria "pulita" per qualche secondo e lo usa come base. In questo modo, il sistema è intelligente e si adatta a ogni ambiente in cui viene posizionato. Siamo consapevoli che **Civic Core** rappresenti solo un punto di partenza. Come ogni prototipo, il sistema è ampiamente **migliorabile**: l'integrazione di sensori più precisi, l'ottimizzazione del codice o l'aggiunta di una connessione Wi-Fi per il controllo remoto via Cloud sono sviluppi che renderebbero il progetto ancora più solido e professionale.

Tuttavia, chiudiamo questo lavoro con grande **soddisfazione**. Per molti di noi, questa è stata la prima vera immersione nel mondo del **maker** e dell'elettronica. Passare dalla teoria dei banchi di scuola a un sistema fisico che 'reagisce' e comunica è stata la sfida più grande. Abbiamo imparato a gestire l'errore, a collaborare in team e a vedere la tecnologia non come un limite, ma come uno strumento per rendere le nostre città più sicure e umane. Civic Core non è solo un circuito, ma la prova che, partendo da zero, si può iniziare a progettare il **futuro**. Abbiamo unito sensori, codice e creatività per un unico obiettivo: muoversi in sicurezza, respirare aria pulita e non aver paura del domani.

Grazie per aver visitato la nostra città del futuro!

I ragazzi dell'I.C. Foscolo-Gabelli - Foggia

CODICI UTILIZZATI: CIVIC CORE

Video: https://canva.link/f4dsthli9c0eezm

1) **ROBOT - BLUETOOTH - POMPA ACQUA**

Questo codice gestisce un robot mobile con sistema di sicurezza antincendio/gas, controllabile tramite Bluetooth.

```
#include <SoftwareSerial.h>

// --- CONFIGURAZIONE PIN ---
SoftwareSerial mySerial(2, 3); // RX | TX per Bluetooth (cavi dai pin 0,1)

const int gasPin = A0;         // Sensore Gas MQ
const int pompaPin = 9;        // Relè Pompa d'acqua

// Pin Motori
const int motorLS_F = 13;      // Sinistro Avanti
const int motorLS_B = 12;      // Sinistro Indietro
```

```
const int motorRS_F = 11;      // Destro Avanti
const int motorRS_B = 10;      // Destro Indietro

// --- PARAMETRI ---
const int sogliaGas = 450;     // Soglia attivazione pompa
int command;

void setup() {
  // Inizializzazione Seriali
  Serial.begin(9600);
  mySerial.begin(9600);

  // Configurazione Output
  pinMode(motorLS_F, OUTPUT);
  pinMode(motorLS_B, OUTPUT);
  pinMode(motorRS_F, OUTPUT);
  pinMode(motorRS_B, OUTPUT);
  pinMode(pompaPin, OUTPUT);

  // Stato iniziale (Tutto spento)
  digitalWrite(pompaPin, HIGH); // Relè spento
  fermaMotori();

  Serial.println("--- SISTEMA ROBOT + GAS AVVIATO ---");
  Serial.println("Pronto a ricevere comandi e monitorare l'aria...");
}

void loop() {
  // 1. MONITORAGGIO GAS (Costante)
  int livelloGas = analogRead(gasPin);

  if (livelloGas > sogliaGas) {
    digitalWrite(pompaPin, LOW); // ATTIVA POMPA
    Serial.print("!!! PERICOLO GAS !!! Livello: ");
    Serial.println(livelloGas);
    mySerial.println("ALLARME: Gas rilevato!"); // Invia avviso allo smartphone
  } else {
    digitalWrite(pompaPin, HIGH); // SPEGNE POMPA
  }

  // 2. CONTROLLO MOVIMENTO (Se arriva segnale Bluetooth)
  if (mySerial.available()) {
    command = mySerial.read();

    // Diagnostica su PC
    Serial.print("Comando BT ricevuto: ");
    Serial.print((char)command);
    Serial.print(" (Codice: ");
```

```
    Serial.print(command);
    Serial.println(")");

    // Logica di Movimento
    if (command == 'F') {
      Serial.println("Stato: Avanti");
      muovi(HIGH, LOW, HIGH, LOW);
    }
    else if (command == 'B') {
      Serial.println("Stato: Indietro");
      muovi(LOW, HIGH, LOW, HIGH);
    }
    else if (command == 'L') {
      Serial.println("Stato: Gira a Sinistra");
      muovi(LOW, LOW, HIGH, LOW);
    }
    else if (command == 'R') {
      Serial.println("Stato: Gira a Destra");
      muovi(HIGH, LOW, LOW, LOW);
    }
    else if (command == 'S') {
      Serial.println("Stato: STOP");
      fermaMotori();
    }
  }

  delay(50); // Piccola pausa per stabilità letture
}

// Funzione di supporto per i motori
void muovi(int sF, int sB, int dF, int dB) {
  digitalWrite(motorLS_F, sF);
  digitalWrite(motorLS_B, sB);
  digitalWrite(motorRS_F, dF);
  digitalWrite(motorRS_B, dB);
}

// Funzione di stop rapido
void fermaMotori() {
  digitalWrite(motorLS_F, LOW);
  digitalWrite(motorLS_B, LOW);
  digitalWrite(motorRS_F, LOW);
  digitalWrite(motorRS_B, LOW);
}
```

2. STAZIONE METEO

Questo codice realizza una centralina di monitoraggio ambientale avanzata con display LCD e sistema di allerta visivo/sonoro.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include "DHT.h"
#include <BH1750.h>

// --- PIN ---
#define DHTPIN 9
#define DHTTYPE DHT11
#define LED_G 5
#define LED_B 6

// --- SOGLIE ---
#define Soglia_Gas  500
#define Soglia_Lux 10

LiquidCrystal_I2C lcd(0x27, 16, 2);
BH1750 lightMeter;
DHT dht(DHTPIN, DHTTYPE);

unsigned long ultimoCambioSchermata = 0;
unsigned long ultimaDiagnostica = 0;
int schermataAttuale = 0;

void setup() {
  Serial.begin(9600);
  Wire.begin();
  lcd.init();
  lcd.backlight();
  dht.begin();
  lightMeter.begin();

  pinMode(Pioggia_DIGITAL, INPUT);
  pinMode(ALARM_PIN, OUTPUT);
  pinMode(LED_R, OUTPUT); pinMode(LED_G, OUTPUT); pinMode(LED_B,
OUTPUT);

  Serial.println(F("======================================"));
  Serial.println(F("   SISTEMA DI MONITORAGGIO AVVIATO    "));
  Serial.println(F("======================================"));
}

void loop() {
  // 1. LETTURA SENSORI
  int gasValore = analogRead(MQ_PIN);
```

```
  float lux = lightMeter.readLightLevel();
  float t = dht.readTemperature();
  float h = dht.readHumidity();
  bool piove = (digitalRead(Pioggia_DIGITAL) == LOW);

  // 2. DIAGNOSTICA SERIALE (Ogni 1000ms = 1 secondo)
  if (millis() - ultimaDiagnostica >= 3000) {
    Serial.println(F("--- REPORT DATI ---"));
    Serial.print(F("Temp: ")); Serial.print(t); Serial.print(F("C | "));
    Serial.print(F("Umid: ")); Serial.print(h); Serial.println(F("%"));
    Serial.print(F("Gas: ")); Serial.print(gasValore);
    Serial.print(gasValore > Soglia_Gas ? F(" [ALLARME]") : F(" [OK]"));
    Serial.print(F(" | Luce: ")); Serial.print((int)lux); Serial.println(F(" lx"));
    Serial.print(F("Pioggia: ")); Serial.println(piove ? F("SI") : F("NO"));
    Serial.println(F("-------------------\n"));
    ultimaDiagnostica = millis();
  }

  // 3. LOGICA SEMAFORO E ALLARME
  aggiornaSemaforoRGB(gasValore);
  gestisciAllarme(lux, gasValore);

  // 4. ROTAZIONE SCHERMATE LCD (Ogni 3 secondi)
  if (millis() - ultimoCambioSchermata >= 3000) {
    aggiornaDisplay(t, h, gasValore, lux, piove);
    ultimoCambioSchermata = millis();
    schermataAttuale = (schermataAttuale + 1) % 3;
  }
}

void aggiornaDisplay(float t, float h, int gas, float lux, bool piove) {
  lcd.clear();
  switch (schermataAttuale) {
    case 0:
      lcd.print(gas > Soglia_Gas ? "!!! GAS ALTO !!!" : "ARIA: OK");
      lcd.setCursor(0, 1); lcd.print("Livello: "); lcd.print(gas);
      break;
    case 1:
      lcd.print("T: "); lcd.print(t); lcd.print(" C");
      lcd.setCursor(0, 1); lcd.print("U: "); lcd.print(h); lcd.print(" %");
      break;
    case 2:
      lcd.print(piove ? "!!! PIOVE !!!" : "METEO: SERENO");
      lcd.setCursor(0, 1); lcd.print("LUCE: "); lcd.print((int)lux); lcd.print(" lx");
      break;
  }
}
```

```
void aggiornaSemaforoRGB(int gasValore) {
  if (gasValore > Soglia_Gas) setColor(255, 0, 0); // ROSSO
  else if (gasValore > (Soglia_Gas / 2)) setColor(255, 80, 0); // ARANCIO
  else setColor(0, 255, 0); // VERDE
}

void setColor(int r, int g, int b) {
  analogWrite(LED_R, r);
  analogWrite(LED_G, g);
  analogWrite(LED_B, b);
}

void gestisciAllarme(float l, int g) {
  if (g > Soglia_Gas || l < Soglia_Lux) digitalWrite(ALARM_PIN, HIGH);
  else digitalWrite(ALARM_PIN, LOW);
}
```

3. APERTURA CANCELLO CON PASSWORD

Questo codice gestisce il controllo accessi di un cancello automatico tramite tecnologia RFID (tessera), dal quale legge il codice identificativo (UID) della tessera.

```
#include <SPI.h>
#include <MFRC522.h>
#include <Stepper.h>

// --- PIN SCHEDA ---
#define SS_PIN 10
#define RST_PIN 9
#define LED 6
#define BUZZ 5

// Bobine motore (Attenzione all'ordine dei pin per l'ULN2003)
Stepper motore(2048, 2, 4, 3, 7);

// Configurazione rapida
const int corsa = 1500;       // Passi per aprire/chiudere
const int pausaVero = 5000;   // Quanto resta aperto (ms)
byte chiaveMaster[] = {0x23, 0xA6, 0xAC, 0x05}; // La mia tessera

MFRC522 rfid(SS_PIN, RST_PIN);

void setup() {
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
```

```
  pinMode(LED, OUTPUT);
  pinMode(BUZZ, OUTPUT);

  motore.setSpeed(12); // Un po' più veloce del default

  Serial.println("--- Sistema Avviato ---");
}

void loop() {
    if (!rfid.PICC_IsNewCardPresent() || !rfid.PICC_ReadCardSerial()) return;

  // Controllo se l'ID è quello giusto
  bool autorizzato = true;
  for (byte i = 0; i < 4; i++) {
    if (rfid.uid.uidByte[i] != chiaveMaster[i]) {
      autorizzato = false;
      break;
    }
  }

  if (autorizzato) {
    Serial.println("Apertura del cancello...");
    suonoOk();

    muovi(corsa); // Apro

    // Effetto attesa con LED che pulsa
    unsigned long inizioPausa = millis();
    while(millis() - inizioPausa < pausaVero) {
      analogWrite(LED, 128 + 127 * cos(2 * PI / 2000 * millis()));
      delay(10);
    }
    digitalWrite(LED, LOW);

    Serial.println("Chiusura automatica");
    muovi(-corsa); // Chiudo

  } else {
    Serial.println("Tessera non valida");
    suonoErrore();
  }

  rfid.PICC_HaltA();
}

// Funzione per muovere il motore
void muovi(int passi) {
```

```
  int stepTotali = abs(passi);
  int verso = (passi > 0) ? 1 : -1;

  for (int x = 0; x < stepTotali; x++) {
    motore.step(verso);

    // Lampeggio blando mentre si muove
    if (x % 100 == 0) digitalWrite(LED, !digitalRead(LED));

    delay(2); // Piccolo delay per la coppia del motore
  }

  digitalWrite(LED, LOW);
  stopMotore(); // per evitare che si scaldi da fermo
}

void stopMotore() {
  digitalWrite(2, LOW); digitalWrite(3, LOW);
  digitalWrite(4, LOW); digitalWrite(7, LOW);
}

void suonoOk() {
  tone(BUZZ, 1200, 150); delay(200);
  tone(BUZZ, 1800, 200);
}

void suonoErrore() {
  tone(BUZZ, 250, 600);
  for(int j=0; j<3; j++) {
    digitalWrite(LED, HIGH); delay(150);
    digitalWrite(LED, LOW); delay(150);
  }
}
```

4. APERTURA CANCELLO CON TASTIERINO PW:

Questo codice gestisce una serratura elettronica intelligente basata su Arduino. Permette di aprire o chiudere una porta tramite un servomotore inserendo un codice numerico su un tastierino.

```
#include <Servo.h>
#include <Keypad.h>
#include <Password.h>
```

```
#define buzzer 11

Servo servo;
bool portaChiusa = false; // Stato iniziale: porta aperta (50 gradi)

Password password = Password("0123");

byte maxPasswordLength = 4;
byte currentPasswordLength = 0;

const byte ROWS = 4;
const byte COLS = 4;

// Mappatura standard (verifica sul Monitor Seriale se i tasti corrispondono)
char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[ROWS] = {2, 3, 4, 5};
byte colPins[COLS] = {6, 7, 8, 9};

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

void setup() {
  Serial.begin(9600);
  pinMode(buzzer, OUTPUT);
  servo.attach(10);

  servo.write(0); // Inizia in posizione APERTA
  Serial.println("Sistema Pronto. Inserisci 0123:");
}

void loop() {
  char key = keypad.getKey();

  if (key != NO_KEY) {
    // Segnale acustico tasto premuto
    digitalWrite(buzzer, HIGH);
    delay(100);
    digitalWrite(buzzer, LOW);

    Serial.print("Tasto: ");
    Serial.println(key);

    if (key == 'C') { // Tasto per resettare se sbagli a digitare
```

```
      resetPassword();
      Serial.println("Password resettata.");
    } else {
      password.append(key);
      currentPasswordLength++;
    }

    // Quando arrivi a 4 cifre, controlla automaticamente
    if (currentPasswordLength == maxPasswordLength) {
      verificaPassword();
    }
  }
}

void verificaPassword() {
  if (password.evaluate()) {
    Serial.println("CORRETTO!");
    okBeep();

    if (portaChiusa) {
      servo.write(180);   // APRE
      portaChiusa = false;
      Serial.println("Porta Aperta");
    } else {
      servo.write(0);  // CHIUDE
      portaChiusa = true;
      Serial.println("Porta Chiusa");
    }
  } else {
    Serial.println("ERRATO!");
    errorBeep();
  }
  resetPassword(); // Pulisce sempre dopo un tentativo
}

void resetPassword() {
  password.reset();
  currentPasswordLength = 0;
}

void okBeep() {
  digitalWrite(buzzer, HIGH);
  delay(600);
  digitalWrite(buzzer, LOW);
}

void errorBeep() {
  for (int i = 0; i < 3; i++) {
```

```
    digitalWrite(buzzer, HIGH);
    delay(150);
    digitalWrite(buzzer, LOW);
    delay(150);
  }
}
```