F.R.E.D. (Fire Retarding Experimental Device)

CMPE3815: Microcontroller Systems

Sam Mason, Owen Schulz

December 11th, 2026

Table of Contents

1. Introduction / Project Overview

This project focused on designing and building F.R.E.D. (Fire Retarding Experimental Device), an autonomous mobile robot that detects a heat source, navigates toward it, and automatically deploys a water sprayer. The purpose was to explore how microcontrollers integrate sensing, movement, and decision-making to create a functional autonomous system inspired by small-scale firefighting robots.

F.R.E.D. uses an infrared heat sensor mounted on a servo to scan its surroundings and determine the direction of the hottest region. The microcontroller processes this information and commands the differential-drive motors to rotate and drive toward that direction. An ultrasonic sensor provides distance feedback so the robot can stop at a safe predetermined range. Once close enough, the robot activates a pump and sprays water toward the target. A manual kill-switch overrides all autonomous behavior for safety.

The goals of the project were to implement reliable heat detection and directional scanning, integrate it with autonomous navigation, control the water-spray mechanism, and operate the entire system from onboard power. By completing these objectives, the project demonstrates how sensing, actuation, and control logic can be combined to create a responsive microcontroller-based robot.

2. System Approach (Conceptual Overview)

We approached the project by organizing the robot into three interacting subsystems: sensing, control, and actuation. The goal was to create a simple but reliable closed-loop architecture where the robot continuously reads the environment, decides what to do, and reacts automatically.

The sensing subsystem uses an infrared heat sensor on a scanning servo to determine the direction of the hottest region, along with an ultrasonic sensor to measure distance to the target. These inputs feed directly into the microcontroller, which runs the decision-making logic. The controller interprets sensor data, determines how to steer the robot, decides when to stop, and triggers the pump once the robot reaches the heat source. A kill-switch overrides all actions for safety.

The actuation subsystem includes the differential-drive motors, the scanning servo, and a relay-driven pump. These parts carry out the controller's commands and form the robot's physical response.

Key design constraints included limited current capacity for the pump, which required a separate power source and relay, and limited mounting space on the RC chassis. We also favored straightforward logic and modular hardware to ensure consistent performance during testing.

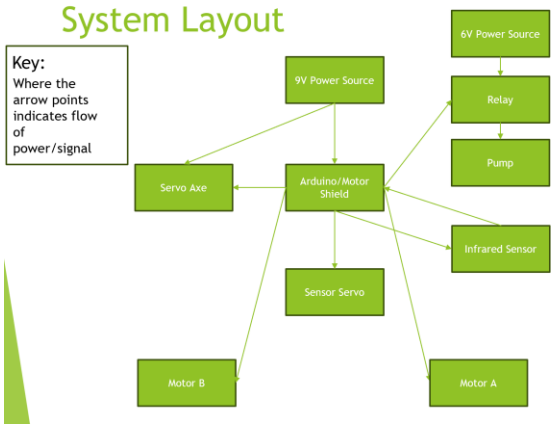

Figure 1 – System Layout

3. Sensing Sub-System

3.1 Description of Sensing Elements

Sensor	Purpose	Concept of Operation	Reason for Selection
Infrared Heat Sensor (analog IR sensor on A2)	Provides a heat-intensity reading used to estimate the direction of the fire.	Outputs an analog voltage proportional to detected infrared radiation. Higher heat results in lower readings. When mounted on a servo and sampled across multiple angles, it identifies the direction of the strongest heat source.	Simple analog interfacing, fast response to temperature changes, and sufficient resolution for angular heat detection.
Ultrasonic Distance Sensor (HC-SR04)	Measures distance to the heat source so the robot knows when to stop, aim, and spray.	Emits a high-frequency pulse and measures echo return time. Distance is computed in ultrasonic_read() using time-of-flight.	Reliable 5–70 cm performance, easy digital interfacing, and essential for triggering close-range behaviors like spraying and AXE movements.
Scanning Servo (servoPin 9)	Rotates the IR sensor across 0°–180° to gather directional heat data.	scanForFire() sweeps the servo in 15° increments, averages several analog readings at each angle, and selects the angle with the lowest (hottest) value.	Low-cost angle scanning without requiring multiple sensors; provides accurate directional detection.

Table 1 – Sensing Elements

3.2 Hardware & Wiring

Component	Arduino Pin	Purpose
IR sensor output	A2	Analog heat reading
Ultrasonic Trigger	12	Sends ultrasonic pulse
Ultrasonic Echo	13	Reads echo duration
Scanning Servo	9	Angle sweep control
AXE Servo	A1	Decorative “axe” movement
Pump Relay	10	Controls pump activation
IR Remote Receiver	3	Decodes remote input

Table 2 - Connection Summary

All sensing components share a common ground with the Arduino. The IR sensor required no additional pull-ups or filtering components. Servo movement required adequate current, so stable battery voltage was important to prevent resets.

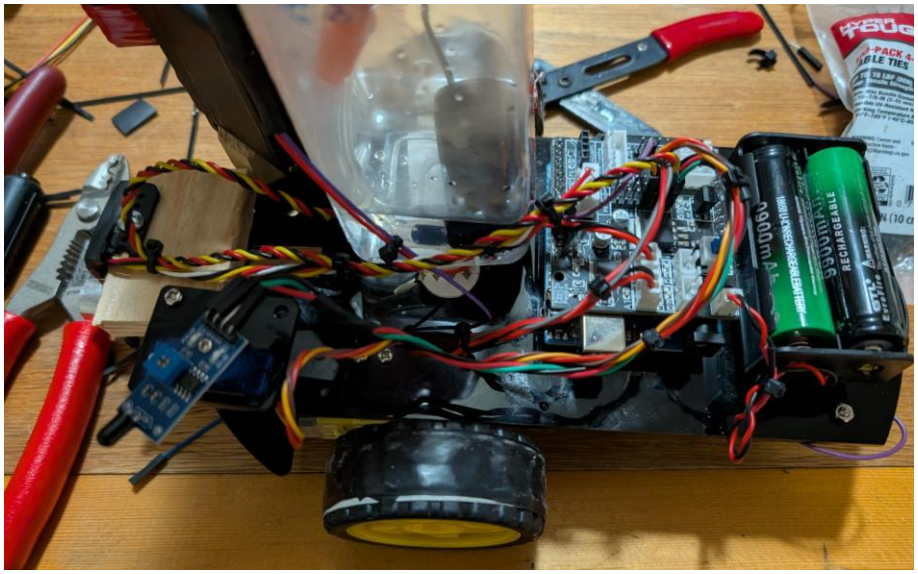

Figure 2 - Sensing components mounted on F.R.E.D.

3.3 Code Snippets for Sensing

Below are the exact sensing functions from your code, shortened only where necessary for clarity.

Ultrasonic Distance Function:

```
float ultrasonic_read() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH, 20000);
  float d = (duration * 0.034) / 2.0;
  return d; // distance in cm
}
```

Directional Heat Scan:

```
void scanForFire(int &bestAngle, float &bestReading) {
 bestReading = 9999;
 bestAngle   = 90;

 float d;

 for (int i = 0; i <= 180; i += 15) {
   scanningServo.write(i);

   delay(i == 0 ? 500 : 200); // stabilize

   d = 0;
   for (int k = 0; k < 3; k++) {
     d += analogRead(heatpin);
   }
   d /= 3.0;

   if (d > 2 && d < bestReading) {
     bestReading = d;
     bestAngle   = i;
   }
 }

 Serial.print("BEST ANGLE = "); Serial.println(bestAngle);
 Serial.print("HEAT READING = "); Serial.println(bestReading);
}
```

Threshold Logic From ThinkFREDThink():

```
if (bestHeat > 0 && bestHeat < sprayHeat) {
    // Fire is close enough to spray
    aimFred();
    spray();
    AXEIT();
} else {
    // Not close enough: move toward best angle
    ChaseFire(bestAngle, bestHeat);
}
```

3.4 Testing Procedures & Results

IR Sensor Testing:

Testing involved placing a heated object at several angles while the servo performed a full sweep.

Results:

- scanForFire() reliably selected the correct direction (±15° resolution).
- Averaging three readings at each angle significantly reduced noise.

Limitations:

- Slowly changing ambient temperature could shift baseline readings.
- Large servo movements momentarily affected analog stability without a settling delay.

Ultrasonic Sensor Testing:

Bench tests used flat targets placed at known distances.

Results:

- Accurate readings in the 10–50 cm range.
- Timeout protection in pulseIn() improved reliability.

Limitations:

- Irregular surfaces sometimes produced inconsistent echoes.

Servo Sweep Testing:

Results:

- Smooth and repeatable motion across the full arc.
- Correct best-angle selection even when heat source was offset.

Limitations:

- Low battery voltage caused occasional resets during combined servo + motor operation.

4. Movement Control Sub-System

4.1 Description of Actuation / Motors

Motor Driver and Drive Motors:
F.R.E.D. uses a differential-drive configuration controlled through a dual H-bridge motor driver. The Arduino sets each motor's direction using digital pins and modulates speed using PWM outputs on pins 5 and 6. All high-level movement—forward, backward, left, right, and stop—is executed through the centralized MotorControl() function. This allows the microcontroller to respond quickly to sensor inputs and adjust movement continuously during autonomous operation.

Scanning Servo:
The scanning servo (pin 9) rotates the infrared heat sensor through a 0°–180° arc. The function scanForFire() commands this sweep in 15° increments and collects heat readings at each position. This servo is essential for determining fire direction and must move reliably and repeatably.

Axe Servo:
A secondary servo connected to pin A1 performs the decorative "AXEIT" sequence once the fire has been extinguished. Although not part of the primary navigation logic, its behavior is still controlled through the movement subsystem and demonstrates additional actuation capability.

Pump and Relay:
The water pump is powered using a separate battery pack and switched using a relay driven by Arduino pin 10. The relay isolates the pump's higher current draw from the microcontroller, preventing brownouts and resets. Pump operation is controlled within the spray() function, which positions the servo nozzle, activates the pump for water deployment, and then returns the nozzle to its resting angle.

4.2 Hardware & Wiring

Component	***Arduino Pin***	***Function***
Left/Right Motor Control	*2, 4*	*Motor direction pins*
Motor PWM Outputs	*5, 6*	*Speed control (analogWrite)*
Scanning Servo	*9*	*IR sensor rotation*
Axe Servo	*A1*	*Axe animation*
Pump Relay	*10*	*Switches pump ON/OFF*
Motor Driver Power	*External battery*	*Drives motors*
Pump Power	*Separate battery*	*Prevents microcontroller resets*

Table 3 - Connection Summary

All actuators share a common ground with the Arduino.
Servos and the relay run from the regulated 5V supply, while the DC motors and pump draw from their respective battery packs to avoid current spikes.

4.3 Code Snippets for Movement Control

Motor Drive Logic:

```
void MotorControl(char direc) {
  switch (direc) {
    case 'F': // forward
      digitalWrite(2, HIGH);
      analogWrite(5, Speed);
      digitalWrite(4, LOW);
      analogWrite(6, Speed);
      break;

    case 'R': // turn right
      digitalWrite(2, HIGH);
      analogWrite(5, Speed);
      digitalWrite(4, HIGH);
      analogWrite(6, Speed);
      break;

    case 'L': // turn left
      digitalWrite(2, LOW);
      analogWrite(5, Speed);
      digitalWrite(4, LOW);
      analogWrite(6, Speed);
      break;

    case 'B': // backward
      digitalWrite(2, LOW);
      analogWrite(5, Speed);
      digitalWrite(4, HIGH);
      analogWrite(6, Speed);
      break;

    case 'K': // kill/stop
      analogWrite(5, 0);
      analogWrite(6, 0);
      break;
  }
}
```

Scanning Servo Sweep (excerpt from scanForFire):

```
for (int i = 0; i <= 180; i += 15) {
  scanningServo.write(i);
  delay(i == 0 ? 500 : 200); // stabilize servo
  float reading = analogRead(heatpin);
}
```

Pump Activation:

```
void spray() {
  AXEServo.write(180);
  delay(1500);
  digitalWrite(pumpin, HIGH);   // pump on
  delay(5000);
  digitalWrite(pumpin, LOW);    // pump off
  AXEServo.write(80);
  delay(1500);
}
```

Kill-Switch Behavior (via IR remote):

```
if (c == '/') {          // kill button
  autoMode = false;
  MotorControl('K');
  Serial.println("AUTO MODE OFF");
}
```

4.4 Testing Procedures & Results

Motor Driver and Mobility Testing:
Each drive direction (F, L, R, B, K) was individually tested using the IR remote before enabling autonomous logic.
Results: Motors responded reliably to PWM changes and direction commands.
Issues: Low battery voltage reduced torque; fresh batteries were required for consistent forward motion.

Servo Testing:
The scanning servo and axe servo were tested for travel range, centering accuracy, and repeatability.
Results: Smooth and predictable motion across the required angles.
Issues: Servo movement caused voltage dips if running from drained batteries.

Pump and Relay Testing:
Pump activation was tested both manually and through the spray() function.
Results: Relay successfully isolated pump current; water spray was consistent during the 5-second activation window.
Issues: Pump load caused resets when accidentally powered from the Arduino supply—resolved by using a dedicated battery pack.

Overall Actuator Performance:
In final integrated testing, all actuators performed reliably, allowing F.R.E.D. to turn toward the fire, drive forward, aim, spray, and perform the post-extinguish AXE sequence without manual intervention.

5. System Integration

5.1 How Sub-Systems Work Together

All subsystems in F.R.E.D. operate in a continuous closed-loop cycle of Sensing → Decision → Action. The sensing subsystem gathers heat and distance data, the control subsystem interprets that information, and the movement subsystem responds accordingly.

Logical Interaction Between Components:

1. *The scanning servo rotates the IR sensor through a 0°–180° sweep.*
2. *The IR sensor samples heat at each angle and sends analog values to the Arduino.*
3. *The ultrasonic sensor measures distance to determine whether the robot is close enough to engage spraying behavior.*
4. *The microcontroller evaluates these inputs inside ThinkFREDThink() to determine the correct action:*
    - *If the robot is not yet close and the IR signal is valid → call ChaseFire().*
    - *When the IR reading indicates the fire is close enough (bestHeat < sprayHeat) → call aimFred() and then spray().*
    - *After spraying, AXEIT() performs the final sequence.*
5. *The motor driver, pump relay, and servos act according to the selected behavior.*

Emergency / Override Logic:

- *At any time, the IR remote's Kill command ('/') disables autonomous mode and immediately stops the motors through MotorControl('K').*
- *This provides a manual override that takes precedence over all autonomous logic.*

Step-by-Step Decision Workflow:

1. *Start autonomous mode using the "Forward Arrow" button.*
2. *Perform heat scan using scanForFire().*
3. *Determine best angle and turn toward it (ChaseFire()).*
4. *Measure distance using ultrasonic_read().*
5. *If too far → continue driving toward the fire.*
6. *If close enough → center the robot (aimFred()), activate pump (spray()), and execute the axe routine (AXEIT()).*
7. *Repeat loop until kill-switch is pressed or fire is "extinguished."*

5.2 Flowchart or Block Diagram

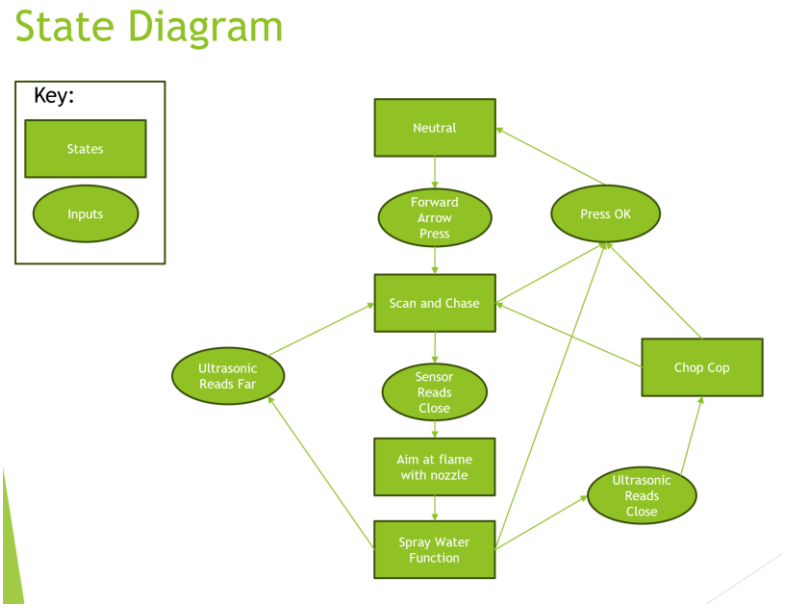

Figure 3 - System Integration Diagram showing Sensing → Decision → Action pipeline with emergency override.

5.3 Integration Testing

Integration testing evaluated how well all subsystems functioned together when running the full autonomous sequence. Testing was conducted in open-floor environments using heated objects as the target.

Test Scenario 1: Full Autonomous Run:

1. *Start autonomous mode.*
2. *Robot performs a full IR sweep and identifies the fire direction.*
3. *Robot turns and drives toward the target.*
4. *When within the distance threshold, robot aims and activates the pump.*

Observation:

- *System reliably centered on the heat source and triggered the spray action.*
- *AXEIT sequence activated as expected.*

Test Scenario 2: Rapid Direction Changes:

The heat source was moved between runs to confirm the robot could adjust.

Observation:

- *The algorithm successfully found the new best angle and reoriented the robot.*
- *Servo delays and averaging prevented erratic behavior.*

Test Scenario 3: Kill-Switch Activation:

Kill button on the IR remote was pressed mid-run.

Observation:

- *Motors stopped immediately.*
- *Autonomous behavior remained disabled until reactivated.*

Adjustments Based on Testing:

- *Added delays to servo movements to reduce noise and improve IR stability.*
- *Implemented averaging of three heat readings per angle to smooth analog inputs.*
- *Added timeout protection to ultrasonic measurement to prevent stall conditions.*
- *Powered pump from a dedicated battery to prevent microcontroller resets during spraying.*

Overall, integration tests demonstrated that the sensing, control, and actuation subsystems operated reliably together, allowing F.R.E.D. to autonomously locate, approach, and "extinguish" a heat source.

6. Performance Commentary

Overall, F.R.E.D. performed reliably and demonstrated successful autonomous fire-targeting behavior. The sensing, control, and movement subsystems worked together smoothly, allowing the robot to detect a heat source, rotate toward it, approach it, and deploy the water sprayer consistently under normal testing conditions.

What Worked Well:

- The IR scanning method proved effective for determining fire direction, especially with the averaging logic that smoothed out analog noise.
- The ultrasonic distance measurement provided dependable detection of close-range targets and consistently triggered the transition from movement to spraying.
- The motor control system responded quickly to direction changes, enabling accurate turns and smooth forward movement.
- The pump and relay setup operated reliably once a separate power supply was used, preventing resets during high-current draw.
- The AXEIT sequence functioned as a fully integrated post-extinguish action, showing that additional actuation could be layered onto the control logic successfully.

Areas for Improvement:

- The IR sensor's analog readings were sensitive to ambient reflections, requiring manual calibration and additional delays for stable measurements.
- Turning accuracy could drift when battery voltage dropped, occasionally causing under- or over-rotation during the chase sequence.
- Servo movement introduced brief voltage dips; dedicated servo power or a stronger voltage regulator would improve reliability.
- The robot's scanning resolution was limited to 15° increments; a finer granularity or a second servo could increase directional accuracy.
-

System Reliability:

When powered with fresh batteries, the system performed consistently and completed the autonomous extinguish sequence multiple times without interruption. Battery sag, however, remained the primary source of inconsistent servo and motor behavior.

Speed, Responsiveness, and Accuracy:

- The robot's responsiveness to heat scans and turn commands was immediate and predictable.
- Forward movement speed was sufficient for indoor demonstrations without overshooting the target.
- Directional accuracy was generally within ±15°, adequate for locating a heat source in a small test environment.
- Pump activation and spraying were highly consistent in timing and behavior.

Environmental Challenges:

Performance varied depending on the surface and ambient conditions. Highly reflective surfaces, irregular shapes, or open spaces sometimes affected ultrasonic readings. Additionally, strong room lighting or background heat sources could influence IR sensor stability. Despite these effects, the robot remained functional and capable of identifying and extinguishing the intended target.

7. Conclusion / Reflection

Building F.R.E.D. provided valuable hands-on experience in integrating sensors, actuators, and control logic into a fully autonomous microcontroller system. Throughout the project, we learned how to break a complex task into coordinated subsystems and ensure that each component—from the IR sensor to the motor driver—worked together reliably. We also gained practical experience in debugging hardware interactions, managing power constraints, and writing modular code that could adapt as the design evolved.

One major takeaway was the importance of system-level thinking. Reliable autonomy required more than just correct code; it depended on stable power delivery, appropriate timing delays, sensor calibration, and predictable mechanical behavior. We saw firsthand how issues such as noise in analog readings or battery voltage drops could influence overall system performance, reinforcing the need for careful testing and iterative refinement.

If given more time, we would explore higher-resolution scanning using a two-axis servo mount, implement filtering algorithms to improve heat detection accuracy, and consider adding obstacle avoidance to make the robot more robust in varied environments. We would also refine the chassis layout to better manage wiring and power distribution.

This project significantly strengthened our engineering skills, particularly in embedded programming, electromechanical integration, and system debugging. It enhanced our ability to design practical solutions, work through real-world limitations, and apply microcontroller concepts to an end-to-end autonomous system. Overall, F.R.E.D. served as a meaningful and rewarding demonstration of how sensing, control, and actuation can be combined to create intelligent robotic behavior.

8. Contribution Statement

This project and report were completed through a collaborative group effort, with both members contributing to every major phase of development. While responsibilities often overlapped, certain tasks were led more strongly by one member based on interest and familiarity.

Project Work Distribution:

- **Coding and Control Logic:** Owen led the development of most of the Arduino code, including the autonomous behavior functions (ThinkFREDThink(), scanForFire(), ChaseFire(), aimFred(), and spray()). Sam participated in debugging, testing, and refining the logic throughout the project.
- **Wiring and Hardware Integration:** Sam led the wiring of sensors, servos, the motor driver, and the relay system, ensuring correct power distribution and stable connections. Owen assisted with hardware troubleshooting and layout decisions.
- **Mechanical Assembly:** Both members worked together on installing the scanning servo, mounting the pump, and integrating the AXEIT mechanism into the chassis.

Documentation Work Distribution:

- **Slide Presentation:** Owen was the primary author and designer of the final project presentation, including the diagrams, layout, and summary slides.
- **Final Report:** Sam led the writing of the final report, organizing the structure, preparing subsystem descriptions, and producing the narrative sections. Owen contributed by supplying accurate technical details, code references, and reviewing content for completeness and correctness.

Overall Collaboration:

Although each member took the lead on different aspects of the project, both actively contributed to all stages of design, implementation, testing, and documentation. The final system and report reflect a balanced partnership in which each person's strengths complemented the other's.

9. Appendix

Complete Code:

```
/*
Owen Schulz
Microcontrollers
12-1-2025

Project: F.R.E.D. (Fire Retarding Experimental Device)
*/

#include <IRremote.h>
#include <Servo.h>

const int ReadPin = 3;
const int echoPin = 13;
const int trigPin = 12;
const int servoPin = 9;
const int heatpin = A2;
const int pumpin = 10;
const int axe = A1;
const int sprayHeat = 150;    // In whatever the IR sensor reads

// Changes throughout code
int Speed = 90;

const unsigned long Forward = 0xB946FF00;
const unsigned long Kill    = 0xBF40FF00;

Servo scanningServo;
Servo AXEServo;

// Declare pinmodes and initiate systems
void setup() {
  Serial.begin(9600);
  IrReceiver.begin(ReadPin, ENABLE_LED_FEEDBACK);

  pinMode(2, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(pumpin, OUTPUT);
  pinMode(trigPin, OUTPUT);
```

```
  pinMode(echoPin, INPUT);

  scanningServo.attach(servoPin);
  AXEServo.attach(axe);
  AXEServo.write(80);
}

// Ultrasonic distance in cm
float ultrasonic_read() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH, 20000); // timeout 20 ms
  float d = (duration * 0.034) / 2.0;           // in cm
  return d;
}

// FRED likes to spread the ashes when he's finished putting the fire out (just
to be safe)
void AXEIT() {
  float read = ultrasonic_read();
  if (read < 20){

    AXEServo.write(180);
    delay(1500);
    AXEServo.write(80);
    delay(1500);
    AXEServo.write(180);
    delay(1500);

    MotorControl('R');
    delay(500);
    MotorControl('L');
    delay(500);
    MotorControl('R');
    delay(500);
    MotorControl('L');
    delay(500);
    MotorControl('K');

    AXEServo.write(80);
    delay(1500);
```

```
    AXEServo.write(180);
    delay(1500);
    AXEServo.write(80);
    delay(1500);
  }
}

// Motor Control commands for FRED's movement
void MotorControl(char direc) {
  switch (direc) {
    case 'F':
      digitalWrite(2, HIGH);
      analogWrite(5, Speed);
      digitalWrite(4, LOW);
      analogWrite(6, Speed);
      break;

    case 'R':
      digitalWrite(2, HIGH);
      analogWrite(5, Speed);
      digitalWrite(4, HIGH);
      analogWrite(6, Speed);
      break;

    case 'L':
      digitalWrite(2, LOW);
      analogWrite(5, Speed);
      digitalWrite(4, LOW);
      analogWrite(6, Speed);
      break;

    case 'B':
      digitalWrite(2, LOW);
      analogWrite(5, Speed);
      digitalWrite(4, HIGH);
      analogWrite(6, Speed);
      break;

    case 'K':
      analogWrite(5, 0);
      analogWrite(6, 0);
      break;
  }
}
```

```
// IR Remote Handling
char remoteDecoder(unsigned long rawdata) {
  switch (rawdata) {
    case Forward: return 'U';
    case Kill:    return '/';
    default:      return '?';
  }
}

// Spraying Water Function
void spray() {
  AXEServo.write(180);
  delay(1500);
  digitalWrite(pumpin, HIGH);
  delay(5000);
  digitalWrite(pumpin, LOW);
  delay(500);
  AXEServo.write(80);
  delay(1500);
}

// Scan for fire: find angle of maximum heat reading
void scanForFire(int &bestAngle, float &bestReading) {
  bestReading = 9999;      // "no fire yet"
  bestAngle   = 90;        // default straight ahead

  float d;

  // scan entire arc 0° → 180° in 15° steps
  for (int i = 0; i <= 180; i += 15) {
    scanningServo.write(i);

    // give servo time to move before reading
    if (i == 0) {
      delay(500);
    } else {
      delay(200);
    }

    // average 3 readings at this angle
    d = 0;
    for (int k = 0; k < 3; k++) {
      d += analogRead(heatpin);
    }
    d /= 3.0;
```

```
    if (d > 2 && d < bestReading) {
      bestReading = d;
      bestAngle   = i;
    }
  }

  Serial.print("BEST ANGLE = ");
  Serial.println(bestAngle);
  Serial.print("HEAT READING = ");
  Serial.println(bestReading);
}

// Chasing Fire (SO brave of FRED)
void ChaseFire(int bestAngle, float bestReading) {
  // turn toward it
  int angleOffset = 90 - bestAngle;  // 90° is "straight ahead"

  // positive offset = right
  // negative offset = left
  Speed = 80;
  if (angleOffset > -8) {
    MotorControl('R');
    delay(abs(angleOffset) * 4);
    Serial.println("Turning Right");
  } else if (angleOffset < 8) {
    MotorControl('L');
    delay(abs(angleOffset) * 4);
    Serial.println("Turning Left");
  }
  MotorControl('K');

  // now drive forward a bit
  Speed = 180;
  MotorControl('F');
  delay(600);
  MotorControl('K');
}

// Aim FRED at the fire (keep turning until roughly centered at 90°)
void aimFred() {
  bool centered = false;

  while (!centered) {
    float closestDist = 9999;
```

```
  int bestAngle = 90;

  // scan entire arc
  for (int i = 0; i <= 180; i += 15) {
    scanningServo.write(i);
    if (i == 0) {
      delay(500);
    }
    delay(200);

    float d = 0;
    for (int k = 0; k < 3; k++) {
      d += analogRead(heatpin);
    }
    d /= 3.0;

    if (d > 2 && d < closestDist) {
      closestDist = d;
      bestAngle = i;
    }
  }

  Serial.print("AIM BEST ANGLE = ");
  Serial.println(bestAngle);
  Serial.print("AIM HEAT READING = ");
  Serial.println(closestDist);

  int angleOffset = 90 - bestAngle;  // 90° is straight ahead

  // If we're already roughly centered, stop
  if (abs(angleOffset) <= 15) {
    centered = true;
    MotorControl('K');
    Serial.println("Centered on fire.");
    break;
  }

  // Otherwise, turn toward it
  Speed = 80;
  if (angleOffset > 15) {
    MotorControl('R');
    delay(abs(angleOffset) * 4);
    Serial.println("AIM: Right");
  }
  if (angleOffset < -15){
```

```
      MotorControl('L');
      delay(abs(angleOffset) * 4);
      Serial.println("AIM: Left");
    }
    MotorControl('K');
  }
}

// When should FRED put out the fire?
void ThinkFREDThink() {
  int bestAngle;
  float bestHeat;

  scanForFire(bestAngle, bestHeat);

  Serial.print("Heat Reading = ");
  Serial.println(bestHeat);

  // Only spray if we have a valid reading and are close enough
  if (bestHeat > 0 && bestHeat < sprayHeat) {
    Serial.println("Close enough: aiming and spraying...");
    aimFred();
    spray();
    delay(200);
    AXEIT();
  } else {
    ChaseFire(bestAngle, bestHeat);
  }
}

bool autoMode = false;

void loop() {

  if (IrReceiver.decode()) {
    char c = remoteDecoder(IrReceiver.decodedIRData.decodedRawData);
    IrReceiver.resume();

    if (c == 'U') {          // forward arrow
      autoMode = true;       // enable autonomous mode
      Serial.println("AUTO MODE ON");
    }
    if (c == '/') {          // kill button
      autoMode = false;      // disable autonomous mode
      MotorControl('K');
```

```
      Serial.println("AUTO MODE OFF");
    }
  }

  if (autoMode) {
    ThinkFREDThink();        // run hunt loop repeatedly
  }
}
```