```
#include <Servo.h>
#include <Wire.h>
#include <math.h>

#define QMC_ADDRESS 0x0D

const int ultrasonic_pin  = 6;
const int L_Motor_pin     = 12;
const int R_Motor_pin     = 13;
const int ARM_SERVO_PIN   = 10;
const int PLATE_SERVO_PIN = 11;

const int S0_PIN  = 3;
const int S1_PIN  = 4;
const int S2_PIN  = 7;
const int S3_PIN  = 8;
const int OUT_PIN = 2;

const int STOPPING_DISTANCE     = 5;
const int SEEKING_DISTANCE_MAX  = 40;
const int PUSH_RESET_MARGIN     = 5;

const int TURN_STRENGTH = 40;
const int L_TURN_OFFSET = 0;
const int R_TURN_OFFSET = 0;

const int L_SCAN_SPEED = 1500 + TURN_STRENGTH + L_TURN_OFFSET;
const int R_SCAN_SPEED = 1500 + TURN_STRENGTH + R_TURN_OFFSET;

const int FORWARD_STRENGTH = 100;
const int FORWARD_OFFSET   = 0;

const int L_FORWARD_SPEED = 1500 - FORWARD_STRENGTH - FORWARD_OFFSET;
const int R_FORWARD_SPEED = 1500 + FORWARD_STRENGTH + FORWARD_OFFSET;

const int BACKWARD_STRENGTH = 100;
const int L_BACKWARD_SPEED  = 1500 + BACKWARD_STRENGTH;
const int R_BACKWARD_SPEED  = 1500 - BACKWARD_STRENGTH;

unsigned long previousMillis    = 0;
const long    interval          = 200;

unsigned long negativeTimer     = 0;
const long    negativeDuration = 500;
```

```
bool isPursuing      = false;
bool hasPushedBlock  = false;
bool isSpinning      = false;

bool wheelsStopped              = true;
unsigned long wheelsStoppedSince = 0;
const unsigned long STARE_TIME   = 1000;

const int ARM_HOME_ANGLE = 170;
const int ARM_PUSH_ANGLE = 10;

const int PLATE_START_ANGLE   = 0;
const int PLATE_ROTATED_ANGLE = 90;

enum Color {
  COLOR_NONE,
  COLOR_RED,
  COLOR_BLUE
};

Color currentColor      = COLOR_NONE;
Color lastValidColor    = COLOR_NONE;
unsigned long lastValidColorTime  = 0;
const unsigned long COLOR_NONE_GRACE = 500;

Color startColor = COLOR_NONE;

float startHeading         = 0.0;
unsigned long runStartTime = 0;
bool   returnPhase         = false;
const float HEADING_TOL    = 15.0;

bool headingAligned        = false;
bool oppositeColorReached  = false;
bool extraDriveDone        = false;
bool finalTurnDone         = false;
bool missionComplete       = false;

unsigned long drivePastOppositeStart = 0;
float finalHeadingTarget   = 0.0;

const unsigned long COMPASS_DELAY_MS = 50000UL;
```

```
Servo L_Motor;
Servo R_Motor;
Servo Arm_Servo;

long current_distance = -1;

long  readDistanceCM(int pin);
void  stopRobot();
void  rotate360();
void  moveForward();
void  moveBackward();
void  pushBlockWithArm();
Color readColor();
void  recoverFromEdge();
void  turn180();
void  markMoving();
void  markStopped();

float getHeadingDegrees();
float normalizeAngle(float a);
float angleDiff(float target, float current);
void  handleReturnPhase();
Color oppositeColor(Color c);

void setup() {
  Wire.begin();

  Wire.beginTransmission(QMC_ADDRESS);
  Wire.write(0x09);
  Wire.write(0x1D);
  Wire.endTransmission();

  L_Motor.attach(L_Motor_pin);
  R_Motor.attach(R_Motor_pin);
  Arm_Servo.attach(ARM_SERVO_PIN);

  {
    Servo plate;
    plate.attach(PLATE_SERVO_PIN);
    plate.write(PLATE_START_ANGLE);
    delay(300);
    plate.write(PLATE_ROTATED_ANGLE);
    delay(300);
    plate.detach();
```

```
  }

  Arm_Servo.write(ARM_HOME_ANGLE);
  delay(400);

  pinMode(S0_PIN, OUTPUT);
  pinMode(S1_PIN, OUTPUT);
  pinMode(S2_PIN, OUTPUT);
  pinMode(S3_PIN, OUTPUT);
  pinMode(OUT_PIN, INPUT);

  digitalWrite(S0_PIN, HIGH);
  digitalWrite(S1_PIN, HIGH);

  startColor         = readColor();
  currentColor       = startColor;
  lastValidColor     = startColor;
  lastValidColorTime = millis();

  float sumHeading = 0.0;
  const int samples = 5;
  for (int i = 0; i < samples; i++) {
    sumHeading += getHeadingDegrees();
    delay(100);
  }
  startHeading = normalizeAngle(sumHeading / samples);

  delay(2000);

  runStartTime = millis();
  stopRobot();
}

void loop() {
  unsigned long currentMillis = millis();

  if (missionComplete) {
    stopRobot();
    return;
  }

  if (!returnPhase && (currentMillis - runStartTime >= COMPASS_DELAY_MS)) {
    Color cNow = readColor();
```

```
    if (startColor != COLOR_NONE && cNow == oppositeColor(startColor)) {
      stopRobot();
      missionComplete = true;
      return;
    }

    returnPhase = true;
    isPursuing  = false;
    stopRobot();
  }

  if (returnPhase) {
    handleReturnPhase();
    return;
  }

  Color rawColor = readColor();
  unsigned long now = currentMillis;

  if (rawColor == COLOR_RED || rawColor == COLOR_BLUE) {
    currentColor       = rawColor;
    lastValidColor     = rawColor;
    lastValidColorTime = now;
  } else {
    if ((now - lastValidColorTime < COLOR_NONE_GRACE) && lastValidColor !=
COLOR_NONE) {
      currentColor = lastValidColor;
    } else {
      currentColor = COLOR_NONE;
    }
  }

  if (!isSpinning && currentColor == COLOR_NONE) {
    recoverFromEdge();
    return;
  }

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    current_distance = readDistanceCM(ultrasonic_pin);
  }

  if (!returnPhase && wheelsStopped &&
      (currentMillis - wheelsStoppedSince >= STARE_TIME) &&
```

```
    !hasPushedBlock) {

  pushBlockWithArm();
  hasPushedBlock = true;
 }

 if (current_distance != -1 && current_distance <= STOPPING_DISTANCE) {
  stopRobot();
  isPursuing = false;

  if (!hasPushedBlock) {
   pushBlockWithArm();
   hasPushedBlock = true;
  }
 }

 else if (current_distance > STOPPING_DISTANCE && current_distance <=
SEEKING_DISTANCE_MAX) {
  negativeTimer = 0;
  isPursuing    = true;
  moveForward();
 }

 else if (current_distance > SEEKING_DISTANCE_MAX || current_distance == -1) {
  if (isPursuing) {
   if (negativeTimer == 0) {
    negativeTimer = currentMillis;
   }

   if (currentMillis - negativeTimer >= negativeDuration) {
    isPursuing    = false;
    negativeTimer = 0;
   } else {
    moveForward();
    return;
   }
  }

  rotate360();
 }

 if (current_distance == -1 ||
   current_distance > STOPPING_DISTANCE + PUSH_RESET_MARGIN) {
  hasPushedBlock = false;
```

```
  }
}

void markMoving() {
  if (wheelsStopped) {
    wheelsStopped = false;
    hasPushedBlock = false;
  }
}

void markStopped() {
  if (!wheelsStopped) {
    wheelsStopped = true;
    wheelsStoppedSince = millis();
  }
}

void stopRobot() {
  L_Motor.writeMicroseconds(1500);
  R_Motor.writeMicroseconds(1500);
  markStopped();
  delay(50);
}

void moveForward() {
  L_Motor.writeMicroseconds(L_FORWARD_SPEED);
  R_Motor.writeMicroseconds(R_FORWARD_SPEED);
  markMoving();
}

void moveBackward() {
  L_Motor.writeMicroseconds(L_BACKWARD_SPEED);
  R_Motor.writeMicroseconds(R_BACKWARD_SPEED);
  markMoving();
}

void rotate360() {
  isSpinning = true;

  L_Motor.writeMicroseconds(L_SCAN_SPEED);
  R_Motor.writeMicroseconds(R_SCAN_SPEED);
  markMoving();

  bool foundTarget = false;
```

```
  unsigned long start = millis();
  const unsigned long maxSpinTime = 3000;

  while (millis() - start < maxSpinTime) {
    long d = readDistanceCM(ultrasonic_pin);
    current_distance = d;

    if (d != -1 && d <= SEEKING_DISTANCE_MAX) {
      foundTarget = true;
      break;
    }

    delay(100);
  }

  stopRobot();
  isSpinning = false;

  if (foundTarget) {
    isPursuing = true;
  }
}

void turn180() {
  unsigned long start = millis();
  isSpinning = true;

  L_Motor.writeMicroseconds(L_SCAN_SPEED);
  R_Motor.writeMicroseconds(L_SCAN_SPEED);
  markMoving();

  while (millis() - start < 800) {}

  stopRobot();
  isSpinning = false;
}

void recoverFromEdge() {
  unsigned long colorStableStart = 0;

  while (true) {
    Color c = readColor();

    if (c == COLOR_NONE) {
```

```
      colorStableStart = 0;
      moveBackward();
    } else {
      if (colorStableStart == 0) {
        colorStableStart = millis();
      }

      if (millis() - colorStableStart >= 1000) {
        stopRobot();
        break;
      }

      moveBackward();
    }
    delay(50);
  }
  turn180();
}

void pushBlockWithArm() {
  int stepDelay = 20;

  for (int pos = ARM_HOME_ANGLE; pos <= ARM_PUSH_ANGLE; pos++) {
    Arm_Servo.write(pos);
    delay(stepDelay);
  }

  delay(50);

  for (int pos = ARM_PUSH_ANGLE; pos >= ARM_HOME_ANGLE; pos--) {
    Arm_Servo.write(pos);
    delay(stepDelay);
  }
}

long readDistanceCM(int pin) {
  pinMode(pin, OUTPUT);
  digitalWrite(pin, LOW);
  delayMicroseconds(2);
  digitalWrite(pin, HIGH);
  delayMicroseconds(5);
  digitalWrite(pin, LOW);

  pinMode(pin, INPUT);
```

```
  long duration = pulseIn(pin, HIGH, 30000);

  if (duration == 0) {
    return -1;
  }

  long distance_cm = duration / 58;

  if (distance_cm <= 0 || distance_cm > 400) {
    return -1;
  }

  return distance_cm;
}

Color readColor() {
  unsigned long r, g, b;

  digitalWrite(S2_PIN, LOW);
  digitalWrite(S3_PIN, LOW);
  delay(20);
  r = pulseIn(OUT_PIN, LOW, 20000);

  digitalWrite(S2_PIN, LOW);
  digitalWrite(S3_PIN, HIGH);
  delay(20);
  b = pulseIn(OUT_PIN, LOW, 20000);

  digitalWrite(S2_PIN, HIGH);
  digitalWrite(S3_PIN, HIGH);
  delay(20);
  g = pulseIn(OUT_PIN, LOW, 20000);

  if (r == 0 || b == 0) {
    return COLOR_NONE;
  }

  bool looksRed  = (10 * r < 7 * b) && (10 * r < 9 * g);
  bool looksBlue = (10 * b < 7 * r) && (10 * b < 9 * g);

  if (looksRed && !looksBlue)  return COLOR_RED;
  if (looksBlue && !looksRed)  return COLOR_BLUE;
  return COLOR_NONE;
}
```

```
float normalizeAngle(float a) {
  while (a < 0)   a += 360.0;
  while (a >=360) a -= 360.0;
  return a;
}

float getHeadingDegrees() {
  int16_t x = 0, y = 0, z = 0;

  Wire.beginTransmission(QMC_ADDRESS);
  Wire.write(0x00);
  Wire.endTransmission();

  Wire.requestFrom(QMC_ADDRESS, 6);
  if (Wire.available() >= 6) {
    uint8_t xL = Wire.read();
    uint8_t xH = Wire.read();
    uint8_t zL = Wire.read();
    uint8_t zH = Wire.read();
    uint8_t yL = Wire.read();
    uint8_t yH = Wire.read();

    x = (int16_t)(xH << 8 | xL);
    z = (int16_t)(zH << 8 | zL);
    y = (int16_t)(yH << 8 | yL);
  }

  float heading = atan2((float)y, (float)x);
  if (heading < 0) heading += 2.0 * PI;

  float headingDeg = heading * 180.0 / PI;
  headingDeg += 180.0;
  return normalizeAngle(headingDeg);
}

float angleDiff(float target, float current) {
  float diff = target - current;
  while (diff < -180.0) diff += 360.0;
  while (diff >  180.0) diff -= 360.0;
  return diff;
}

Color oppositeColor(Color c) {
```

```
  if (c == COLOR_RED)  return COLOR_BLUE;
  if (c == COLOR_BLUE) return COLOR_RED;
  return COLOR_NONE;
}

void handleReturnPhase() {
  if (!headingAligned) {
    float heading = normalizeAngle(getHeadingDegrees());
    float diff    = angleDiff(startHeading, heading);
    float adiff   = fabs(diff);

    if (adiff > HEADING_TOL) {
      if (diff > 0) {
        L_Motor.writeMicroseconds(L_SCAN_SPEED);
        R_Motor.writeMicroseconds(L_SCAN_SPEED);
      } else {
        L_Motor.writeMicroseconds(R_SCAN_SPEED);
        R_Motor.writeMicroseconds(R_SCAN_SPEED);
      }
      markMoving();
      return;
    } else {
      stopRobot();
      headingAligned = true;
      finalHeadingTarget = normalizeAngle(startHeading + 90.0);
      return;
    }
  }

  if (!oppositeColorReached) {
    moveForward();
    Color c = readColor();
    if (c != COLOR_NONE && startColor != COLOR_NONE && c == oppositeColor(startColor)) {
      oppositeColorReached   = true;
      drivePastOppositeStart = millis();
    }
    return;
  }

  if (!extraDriveDone) {
    if (millis() - drivePastOppositeStart < 1000UL) {
      moveForward();
      return;
    } else {
```

```
      extraDriveDone = true;
      stopRobot();
    }
  }

  if (!finalTurnDone) {
    float heading = normalizeAngle(getHeadingDegrees());
    float diff    = angleDiff(finalHeadingTarget, heading);
    float adiff   = fabs(diff);

    if (adiff > HEADING_TOL) {
      if (diff > 0) {
        L_Motor.writeMicroseconds(L_SCAN_SPEED);
        R_Motor.writeMicroseconds(L_SCAN_SPEED);
      } else {
        L_Motor.writeMicroseconds(R_SCAN_SPEED);
        R_Motor.writeMicroseconds(R_SCAN_SPEED);
      }
      markMoving();
      return;
    } else {
      stopRobot();
      finalTurnDone   = true;
      missionComplete = true;
      return;
    }
  }

  stopRobot();
}
```