Robotics Culminating Report

Ryan Jin, Aleksa Misic, Achita Anantachina

January 2024

1 Research and Problem Definition

The technologies employed in CETA Robotics are applicable to a wide array of real-world challenges. For instance, any autonomous driving system necessitates an algorithm that adheres to a predetermined guideline. In this context, our "line following robot" could be adapted for applications such as road-following autonomous vehicles.

2 Design Proposal

The Toronto CETA Robotics competition outlines a challenge wherein our robot is required to traverse a line track multiple times before reaching its final destination and subsequently idling. The competition imposes several constraints, which are enumerated as follows:

1. The power supply is restricted to 4 AA batteries exclusively. However, the controller (Raspberry Pi Pico) may be powered by an alternative source.
2. The wheels must be actuated by the FM90 DC Motor.
    - While other motors available from cool-mcu.com are permissible, they must remain unmodified.
    - The appropriate wheels for the FM90 DC Motor are the FS90R-W Wheels.
3. It is recommended that competitors fabricate their own chassis within the prescribed dimensions.
    - The robot's footprint (i.e., the maximum width and length) must not exceed 12 cm by 18 cm.
        - A more compact robot enhances maneuverability.
4. Teams may deploy multiple robots for distinct challenges; however, adherence to the size restrictions remains mandatory for each unit.

Our team has elected to utilize a single robot to address all three challenges. Robot 17119, representing our team's entry, will integrate components provided in the Cool MCU robot kit, in addition to an ultrasonic sensor (sourced from Achita's Basement) mounted on a custom 3D-printed chassis.

3 Prototyping and Construction

The initial prototype was assembled using the components from the Cool MCU robot kit. The construction process is outlined below:

1. **Circuit Assembly:** Develop the primary circuit, incorporating the Raspberry Pi Pico, two manual reset buttons, and a potentiometer.
2. **Chassis Assembly:** Construct the main chassis, which is divided into two principal sections:
    - The lower section of the chassis accommodates the motor control circuitry and the DC motors. The front edge of this section also supports three optical sensors that interface with the main circuit.
    - A raised platform is integrated to house the central processing circuitry.
3. **Final Assembly:** Install the unpowered front wheel.

After evaluating the spatial utilization and component placement in the prototype, a custom chassis was designed using Fusion 360, and its modular components were fabricated via 3D printing.

4 Programming

The objective of our program is to develop an autonomous robot capable of following a designated track marked by a black line, using three optical sensors. These sensors produce readings based on the intensity of reflected light, yielding values from 0 to 1000. Rather than employing the template code provided by CoolMCU, we have developed our own software solution.

4.1 Sensor Data Acquisition and Helper Functions

1. **Built-in Functions:** The function `analogRead(Sensor)` is used to obtain data from the optical sensors.
2. **Custom Helper Functions:**
   - **scaling():** To normalize the sensor values, the following function is implemented:

```
int scaling(float pin){
  return round(analogRead(pin) / 10) * 10;
}
```

   - **runMovement():** This function simplifies wheel control by setting the speeds of the right and left servos, respectively:

```
void runMovement(int rightSpeed, int leftSpeed){
  rightServo.write(rightSpeed);
  leftServo.write(leftSpeed);
}
```

   - **stopMoving():** As implied by its name, this function halts both wheels:

```
void stopMoving(){
  rightServo.write(94);
  leftServo.write(94);
}
```

   - **forwardState():** This helper function verifies whether the sensor readings (left, middle, and right) fall within the desired thresholds. It returns `true` if all conditions are satisfied:

```
bool forwardState(int midL, int midH, int leftL, int leftH, int rightL
    , int rightH){
  bool a, b, c, d, e, f;
  a = midL <= scaling(midSensor);
  b = scaling(midSensor) < midH;

  c = leftL <= scaling(leftSensor);
  d = scaling(leftSensor) < leftH;

  e = rightL <= scaling(rightSensor);
  f = scaling(rightSensor) < rightH;

  return a && b && c && d && e && f;
}
```

- **endingDetection():** This function detects the conclusion of the track by monitoring an all-white sensor state through `forwardState()`. To mitigate false positives during normal operation, the function validates that the all-white condition persists over a defined timeframe:

```
void endingDetection(){
  if (count == 0 and forwardState(0, 700, 0, 700, 0, 700)){
    firstCondition = true;
    count++;
  }
  else if (count > 0 && forwardState(0, 700, 0, 700, 0, 700)){
    count++;
  }
  else{
    firstCondition = false;
    secondCondition = false;
    count = 0;
  }

  if (count > 24 && forwardState(0, 700, 0, 700, 0, 700)){
    secondCondition = true;
  }
}

```

3. **PID Control Implementation:**

   To achieve autonomous line following, we employ a PID control system. The robot continually adjusts its trajectory by processing an error signal and computing a control signal, denoted as the PID control variable. The PID control is defined by:

$$u(t) = K_p e(t) + K_i \int e(t)\,dt + K_d \frac{de(t)}{dt}$$

   where:

   - $u(t)$ is the control signal.
   - $e(t)$ is the error between the set point and the current position, defined as:

$$e(t) = \text{Setpoint} - \text{CurrentLocation}$$

   - K_p, K_i, and K_d are the proportional, integral, and derivative gains, respectively.
   - **Proportional Gain:** This term reacts immediately to the error, computed as:

$$P = K_p e(t)$$

     A high K_p may cause overshooting, while a low K_p may result in insufficient corrective action.
   - **Integral Gain:** This term accumulates past errors to mitigate any persistent steady-state error:

$$I = K_i \int e(t)\,dt$$

     The constant K_i determines the aggressiveness of this correction.
   - **Derivative Gain:** The derivative term predicts future errors based on the current rate of change, thereby smoothing the robot's response:

$$D = K_d \frac{de(t)}{dt}$$

     The constant K_d scales the damping effect to reduce overshooting.

Defining the Error: Given the limited information from the three binary optical sensors, we define the error as the difference between the readings of the right and left sensors, scaled appropriately:

```
error = round((scaling(rightSensor) - scaling(leftSensor)) / 100);

```

The individual PID components are computed as follows:

```
int P, I, D;
P = error;
I += error;
D = error - prevError;
int PIDval = (Kp * P) + (Ki * I) + (Kd * D);
delay(dT);

```

PID Control Routine: The following function represents one iteration of the PID correction process:

```
void PID_Control()  {

  // Compute the error based on sensor differences
  error = round((scaling(rightSensor) - scaling(leftSensor)) / 100);

  P = error;              // Proportional component
  I += error;             // Integral component
  D = error - prevError; // Derivative component

  int PIDval = (Kp * P) + (Ki * I) + (Kd * D);  // Control signal

  // Update the previous error
  prevError = error;

  // Adjust wheel speeds accordingly
  rightWheel = neutralSpeed + PIDval;
  leftWheel = neutralSpeed - PIDval;
}

```

A continuous control loop is implemented to constantly correct the robot's trajectory:

```
void followLine(){
  bool runLineFollow = false;
  while (!runLineFollow){

    PID_Control();
    runMovement(rightWheel, leftWheel);

    endingDetection();

    runLineFollow = firstCondition && secondCondition;
    delay(dT);
  }
}

```

In summary, our control system leverages a PID controller to realize an autonomous, track-following robot. Through the application of control theory principles, we have developed a robust and efficient algorithm that meets our design objectives.

5 Testing and Debugging

During testing, four critical variables were optimized to enhance the robot's performance:

1. **Objective:** The primary objective is to minimize the time required for the robot to complete the track. This is achieved by incrementally optimizing the PID constants at a given speed, then slightly increasing the speed, and subsequently fine-tuning the constants.
2. K_p **(Proportional Gain):** This constant directly influences the robot's turning response:
   (a) An excessive K_p value results in overcorrection, leading to excessive turning.
   (b) An insufficient K_p value fails to produce an adequate turn, preventing the robot from realigning with the track.
3. K_i **(Integral Gain):** This term corrects for persistent steady-state errors. To evaluate K_i, the robot is subjected to slight disturbances:
   (a) If the robot overcorrects, K_i is too high.
   (b) If the robot is sluggish in realigning, K_i is too low.
4. K_d **(Derivative Gain):** This constant modulates the smoothness of the robot's corrective actions:
   (a) A low K_d value results in abrupt, sharp turns.
   (b) A high K_d value causes the robot to react too slowly.

The calibration process is conducted sequentially:

1. Initialize all constants to zero.
2. Determine the optimal K_p by testing successive integer values until an overshoot is detected, thereby establishing an interval $[n-1, n)$ for the optimal value.
3. Utilize a binary search methodology within this interval:
   (a) Compute the midpoint $\frac{a+b}{2}$ of the current interval.
   (b) Adjust the interval based on whether the midpoint produces an excessive or insufficient response.
4. Repeat this process for K_i and K_d until optimal performance is achieved.

This methodology, akin to performing a binary search on a continuous interval, ensures that optimal constants are determined in an efficient manner.

6 Robot Demonstration

The demonstration video illustrates the successful integration of our program with the hardware, thereby meeting the requirements of the CETA Robotics competition. Click here to view the demonstration.