Coding Types Vs. Time and Accuracy

New Mexico Supercomputing Challenge

Final Report
April 7, 2021

Team 39
Media Arts Collaborative Charter School

Team Members
Everett Raucci

Mentors
Vary Coates
Creighton Edington

Coding Types Vs. Time and Accuracy

Everett Raucci

---

1. Executive Summary

The purpose of this experiment is to find the most efficient code that makes the robot identify and stop at the blue block. I wrote four codes, each designed to locate and drive to a colored object. The robot would stop, and that would be the end of the code. My first code was the left-right. This code would make the robot turn either left or right depending on where the colored object was. My second code was the left-center-right, which would turn left, right, or drive forward depending on where the colored object was on the screen. My next code was the Brute Force, because there are over 900 lines of code. Brute Force tells the motors to change their percentage of power by 1 for every one increment of change in the horizontal position of the target on the screen. The last code I wrote was the Linear Function. This code tells the robot to turn a certain amount either left or right based on the output of a linear function with the input being the horizontal location of the target value on the screen.

---

2. Introduction

This could be applied to the real world. For example we are needing robots more and more as time progresses. One thing we could use is robots that can recognise color in, say, a loading facility. The robot could identify color-coded items and move them to different places. I wrote four different codes for my Wallaby-controlled robot. A camera on the front of the robot sends images to the robot giving it the location of the target along a horizontal axis. The camera provides an x-axis value for the target location, and each code uses those values to tell the motors what percentage of power to use.

The robot also has an infrared sensor for detecting how close the object is by sending out an infrared beam. After the beam bounces off whatever is in front of it, the beam returns to the sensor with a value corresponding to the strength of the beam, indicating how far away the object is.

The robot has two wheels, and a ball bearing on the back for support. There are two motors, one for each wheel. The motors are activated when the code starts.

I used the KISS* Institute for Practical Robotics (KIPR) software suite, and I wrote the codes in C language. The first code is called Left-Right, which makes the robot turn left or right toward the block depending on where it is on the screen. The next code I wrote is called the Left-Right-Center. Again, this turned left, right, or drove forward toward the block, depending on the location of the target on the screen.

The next is called Brute Force because there are over 900 lines of code. Brute Force tells the motors to change their percentage of power by 1 for every one increment of change in the horizontal position of the target on the screen.

The last code is the Linear Function. This code tells the robot to turn a certain amount either left or right based on the output of a linear function with the input being the horizontal location of the target value on the screen.

3. Hypotheses

H0 (null hypothesis): All four algorithms will perform the same as the “Basic Left-Right Algorithm”.

H1 (negative hypothesis): None of the four algorithms will perform better than the “Basic Left-Right Algorithm”.

H2 (positive hypothesis): All of the four algorithms will perform better than the “Basic Left-Right Algorithm”.

H3 (mixed hypothesis): At least one of the four algorithms will perform better than the “Basic Left-Right Algorithm”.

4. Methods

I used a basic Left-Right algorithm to compare the other algorithms against (see Image 1).

```
if (x <= 80) //if target is on the left half of the screen
{
  motor(0,60); // left motor at 60% power
  motor(3,20); // right motor at 20% power
}
if(x > 80)
{
  motor(0,20);
  motor(3,60);
}
```

Image 1. Left-Right algorithm

I ran the Left-Right algorithm robot code 15 times. However, my reaction time to the robot stopping wasn't precise enough. To address this problem, I added code to use the internal clock of the robot to record how much time it took to get to the target. This allowed me to record the value within a hundredth of a second. I redid the Left-Right algorithm times and the misalignment of the robot from the center of the target.

I then ran the Left-Center-Right algorithm (see Image 2) 15 times, testing the time the robot took to get to the target and recorded its misalignment to the center of the target.

```
if (x <= 32) // extreme left (far left ⅛ of screen)
{
  motor(0,8);
  motor(3,72);
}

if (x > 32 && x <= 64) // left ( ⅜ part of the screen)
{
  motor(0,24);
  motor(3,56);
}

if(x > 64 && x < 96) // object center (⅜ part of the screen)
{
  motor(0,40);
  motor(3,40);
}
if (x >= 96 && x < 128) // right (⅝ part of the screen)
{
  motor(0,56);
  motor(3,24);
}

if (x >= 128) // extreme right ( 5/5 part of the screen)
{
  motor(0,72);
  motor(3,8);
}
```

Image 2. Left-Center-Right algorithm.

I then ran the Brute Force code (see Image 3, 4, and 5) 15 times, testing the time the robot took to get to the target and recorded its misalignment to the center of the target.

```
if (x == 0)
{
   motor(0,0);
   motor(3,80);
}
if(x == 1)
{
   motor(0,1);
   motor(3,80);
}

if(x == 2)
{
   motor(0,1);
   motor(3,79);
}
```

Image 3. Brute Force algorithm beginning

```
if(x == 78)
{
   motor(0,39);
   motor(3,41);
}

if(x == 79)
{
   motor(0,40);
   motor(3,41);
}

if(x == 80)
{
   motor(0,40);
   motor(3,40);
}
```

Image 4. Brute Force algorithm middle

```
if(x == 158)
    {
        motor(0,79);
        motor(3,1);
    }

if(x == 159)
    {
        motor(0,80);
        motor(3,1);
    }

if(x == 160)
    {
        motor(0,80);
        motor(3,0);
    }
```

Image 5. Brute Force algorithm end

I then ran the Linear Function code (see Image 6) 15 times, testing the time the robot took to get to the target and recorded its misalignment to the center of the target.

```
x = get_object_center_x(0,0);
motor0 = (-1 * (.5 * x)) + 80;
motor3 = .5 * x;
 motor(0,motor0);
 motor(3,motor3);
```

I tested the codes each 15 times, and then averaged the individual code results (time and accuracy). The robot traveled 136 centimeters on a white tablecloth during each test, and each time at the start of the test, the robot started at a set point on the cloth, which stayed the same throughout the 15 tests. I used white because it has no color like blue to confuse the robot. The robot is stopped by the code when the infrared sensor returns the value 370. This meant that the robot would stop a certain distance away. After the robot got to the target, I would measure in centimeters how far off it was from the center of the block (accuracy). After 15 tests, I averaged the individual code results. I also recorded the minimum and maximum length of times and how close it was to the center of the target for a tiebreaker.

5. Results

I measured the accuracy of the algorithms by measuring (in centimeters) how far off the center of the camera was to the target. The Left-Right algorithm was the least accurate, with the Left-Center-Right being slightly better. The Brute Force and Linear Functions algorithms are about equal in terms of accuracy.. (See figure 1.)

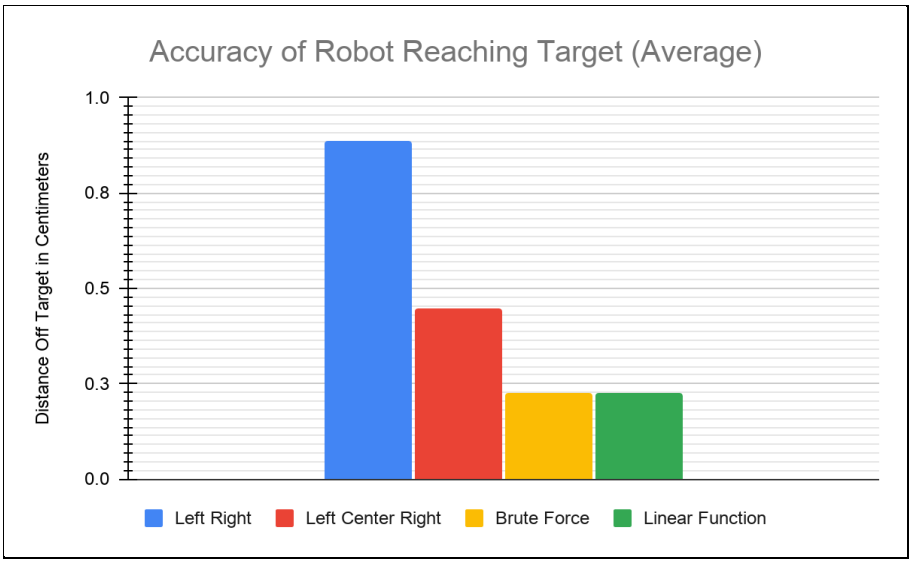

Figure 1

I used the internal clock of the robot to measure how long the robot took to get to the target. The time was measured down to a hundredth of a second. The Left-Right algorithm took the most time to get to the target, with the Left-Center-Right and Brute Force with equal times better than the Left-Right. The Linear Function was the most accurate, but by a small fraction of a second. (See figure 2)

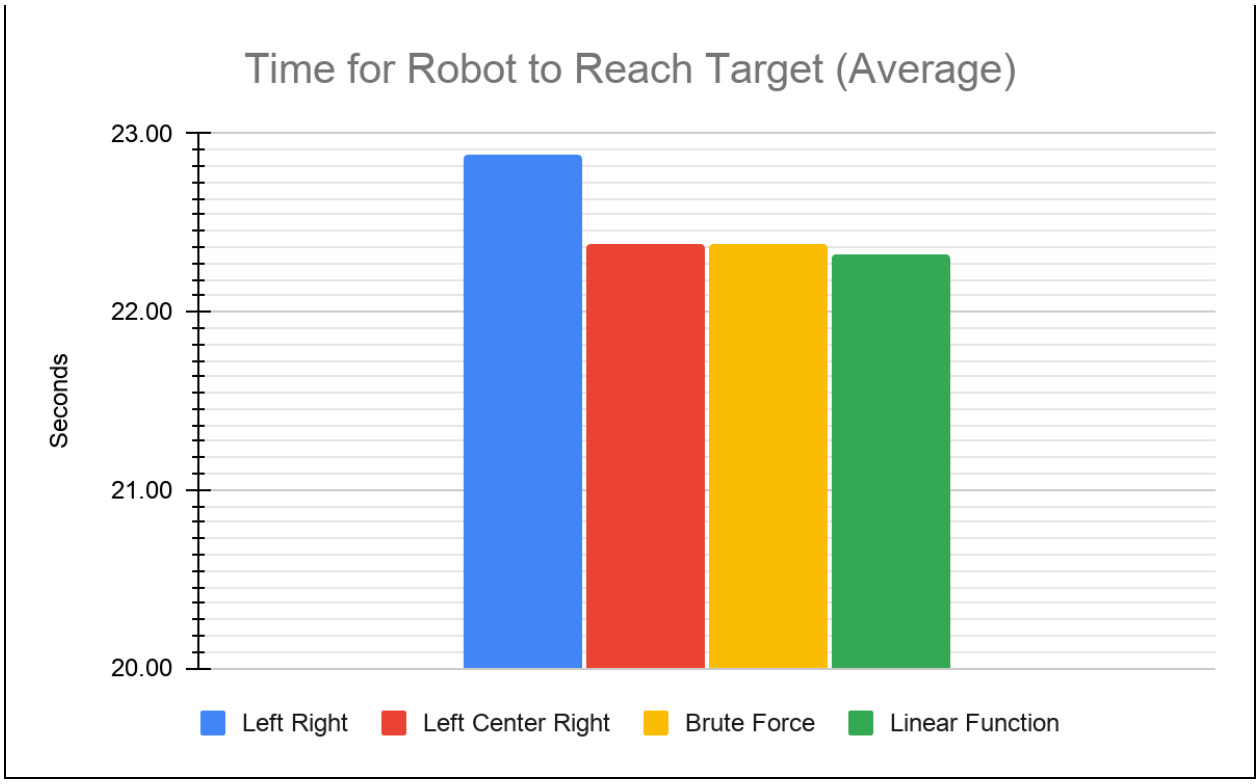

Figure 2

The results I found was that the Linear Function is the most efficient and accurate out of the four codes, Brute Force being a close second. This supports my last hypothesis, that at least one of three algorithms will work better than my Left-right algorithm.

6. Acknowledgements

I want to thank Mr. Edington, who without I would have a much harder time completing this project. He helped me stay on track and helped me understand how to enter the challenge. I also would like to thank Vary Coates, who gave me feedback on all versions of my report, and gave me class time to work on it.