object tracking

Using Bangbang, pcontrol, pdcontrol and pidcontrol



Alex Waweru

Leon

Ayomide

**INTRODUCTION**

The task was decided into two parts as outlined below:

1. Implement various kinds of feedback control to enable a robot to maintain a fixed distance from an obstacle. For each type of control, experiment with different gains and record your observations of the behavior of the robot as the gain changes. For each type of feedback control, record very short videos of the behavior of your robot for at least 3 values of the gain (too high, too low, and “just right”).
2. Have your robot implement feedback control to track and follow a colored object. You must use a camera. You can, in addition, use an ultrasonic sensor if you wish. You have a choice of using P, PD or PID control, but you shouldn’t use bang-bang control.

**APPROACH**

**Part 1**

Error was calculated as the difference between the distance measured by the ultrasonic sensor and the distance to be maintained.

Output is the speed of both motors.

The ultrasonic sensor was used to get the distance between the object and the robot.

1. **Bang-bang control:**

The following algorithm was used to control the speed of the motors:

K = 50

If error > des\_distance:

Output = k \* (+1) + 0

If error < des\_distance:

Output = k \* (-1) + 0

1. **P-control:**

The following algorithm was used to control the speed of the motors:

K = 50

If error != des\_distance:

Output = k \* error + 0

1. **PD-control:**

The following algorithm was used to control the speed of the motors:

Previous\_error = 0

Error = distance – des\_distance

Previous\_error = error  
K = 50

If error != des\_distance:

Output = k \* error + (error – previous\_error) + 0

1. **PID-control:**   
   The following algorithm was used to control the speed of the motors:

#this two are initiated outside the while loop

errors = []

Previous\_error = 0

#this are initiated inside the while loop

Error = distance – des\_distance

Error.append(error)

#Calculate the integral of error(sum of errors)

sum\_of\_errors = reduce(lambda x, y: x+y, errors)

Previous\_error = error  
K = 50

If error != des\_distance:

Output = k \* error + (error – previous\_error) + sum\_of\_errors + 0

**Part 2**

The solution contains two controls, one for turning towards the colored object and the other for maintaining a distance between the colored object and the robot.

The camera and ultrasonic sensor were used to get horizontal position of the object and the distance between the object and the robot (measured in centimeters) respectively.

1. **Distance Control:**

The pcontrol structure (as implemented below) was used to control the distance between the colored object and the robot.

K = 50

If error != des\_distance:

Output = k \* error + 0

1. **Turning Control:**

The pcontrol structure (as implemented below) was used to control the left and right turning of the robot towards the colored object.

Error represents the difference between the half the width of the camera and half the width of the largest colored object.

error = CAMERA\_WIDTH\_PIXELS/2 - w/2

K = 50

Output = k \* error + 0

If (x < error):

leftMotor.run\_forever(speed\_sp = output)

rightMotor.run\_forever(speed\_sp = 0)

if (x>error):

rightMotor.run\_forever(speed\_sp = output)

leftMotor.run\_forever(speed\_sp = 0)

**RESULTS**

From the experiemnt the following qualitative results were observed:

1. The **bangbang control** makes the robot follow the object at a steady speed regardless of how close or far the object is from the robot. The robot hurls back and forth roughly depending on how high the contant is.
2. With the **pcontrol**, the speed at which the robot follows the object (capped at 990) depends on the distance between the robot and the object. The smaller the error (difference between desired distance and actual distance) the lower the speed. Unlike in the bangbang control, in pcontrol the robot hurls back and forth less vigorously.
3. With the **pdcontrol**, the speed at which the robot follows the object (capped at 990) depends on the distance between the robot and the object, and the difference between the previous error and the current error. The smaller the error (difference between desired distance and actual distance) the lower the speed. Unlike in the bangbang control, in pcontrol the robot hurls back and forth less vigorously. Moreover, the robot is able to achieve the desired distance when the object is at rest.
4. With the **pidcontrol**, the speed at which the robot follows the object (capped at 990) depends on the distance between the robot and the object, and the difference between the previous error and the current error, and the sum of all previuos errors. The smaller the error (difference between desired distance and actual distance) the lower the speed. Unlike in the bangbang control, in pcontrol the robot hurls back and forth less vigorously. Moreover, the robot is able to achieve the desired distance when the object is at rest.
5. In part 2, with **turning towards the colored object** the **pcontrol** was used. The speed at which the robot turns (either left or right) depends on how far from the centre of the screen the centre of the largest colored object is. The larger the difference, the faster the robot turns.

**CONCLUSION**

The smoothest control architecture was pid control. The robot moved smoothly and swiftly, and could achieve the desired distance even when the object was in motion.

In the future, the team would explore using run-time autotuning of a robot controller using a genetics based machine learning control scheme. This would help reduce non linear disturbances and also achieve an optimal speed constant.