ENPH 353 LogBook

Name: Yousif El-Wishahy

Lab 1 - Jan 13th 2022

Worked on installing lubuntu on usb flash drive, got the .iso file and flashed it to usb with balenaetcher. One problem was that it ran pretty slow - will have to consider this for next time. Other than that, I watched a video about basic linux terminal commands, ran the gazebo simulation and noted the real-time factor (im assuming this related to performance). Linux was slow but otherwise worked.

Lab 2 - Jan 20 2022

Attempting daul boot and vm for faster lubuntu speeds

The usb linux installing is too slow, I'm going to try to daul boot it on a hard drive partition on my laptop instead.

Update: After 2 hours of troubleshooting , I could not manage to get things to work. I got to the stage of clearing a partition for the linux installing on the drive. But when I booted from the usb i flashed the linux .iso to, there are no installing options as almost every online source says there should be. Initially, I could not even boot to the grub menu and had disabled fast boot in the bios.

Update 2: I decided to go with a vm on my desktop instead. I installed all the required files and downloaded the iso and ran the conversion command - note that i had to modify the command for directory and file name:

```
VBoxManage convertfromraw ~/Downloads/lubuntu_18.04_21-01-18_UEFI.iso
lubunt_ENPH353_v2.vmdk --format vmdk
```

From there things went well until I created the vm from the converted .iso file. I enabled the uefi setting in the vm and increased ram allocation, but when the vm booted it only went into the uefi shell. After some online research, turns out i had to modify the startup.nsh file to include '\EFI\ubuntu\grubx64.efi ' and then things worked. From there the lubuntu vm worked and was connected to the web. Notably, it ran MUCH better in vm than off usb but that might be attributed to the 24gb ram or 12 core cpu I have on my desktop :) - I suspect the main reason though, is data transfer speeds of usb.

Actual lab 2: programming line detection

This lab consisted of two learning goals for me:

1. Understanding how to utilize OpenCV
2. Implementing line following using OpenCV and numpy

I've worked with python, jupyter and numpy a lot during the past year or two in personal projects, courses and my internship where I process radar signals using low pass filters and such.

Anyways, here's what I am try to achieve: **There is a video containing a traversal of a path, we want to make a ball follow this path using numpy and OpenCV**

Breaking it down:

- Split the video into frames and process each frame
- Draw a circle onto frame to show 'ball' following path, no need for 3d animations
- Detect path to place circle roughly at the centre of the path
- This should make it look like the ball is following the path

OpenCV does the heavy lifting for converting the video into an array of frames for me to process, from there I was able to use numpy to look for the location of all pixels containing the colour of the path. To get the cartesian image coordinates of where the ball should be, I just averages all the x coordinates and y coordinates of the points where the path color appeared. Then use OpenCV to draw the circle at that averaged point.

This algorithm worked well, and the task was complete.

Considerations for further improvements:

- the algorithm doesn't have to search the whole frame for the path if we know its usually closer to the bottom for example
- the color to detect was manually inputted, is there a way to automate this?
- could edge detection be a better method instead of averaging color coordinates?

Lab 2 results demo

https://drive.google.com/file/d/1XesD-TIiiiouQgJ0daAhKpeUp1_MtAX-/view?usp=sharing

Lab 3 Jan 27

Dual booting ubuntu on laptop instead

The vm was still too slow, so i dual booted ubuntu as i couldn't get the lubuntu .iso to install on a partition. So I dual booted latest ubuntu and installled all required packages and it ran much smoother for the gazebo simulation!

Self driving robot with computer vision and PID in gazebo/ROS

Goal: Simulate a robot in a ROS and gazebo environment and make the robot drive and follow a path by implementing computer vision and control software.

Tools/skills applied and learned:

- Linux environment
- ROS (robot operating system)
- Gazebo (Simulation world)
- Python
- OpenCV and computer vision
- Control theory (PID?)

The first portion of the lab consisted of building and running the gazebo simulation world and robot. The robot and world setup in ros uses an xml style code where components of the world or robot are added in xml with features and textures.

Next external plugins are supported that move the robot and prove camera feeds.

I installed these plugins:

1. skid steer controller
2. camera

The skid steer controller essentially controls the robot to turn as a skid steer vehicle would, with forward driving and skid steering. The control works by publishing to the robot through command line or python script. The publishing works something like this:

```
#!/usr/bin/env python

#imports
import rospy
from geometry_msgs.msg import Twist

#register publisher to ros simulation
rospy.init_node('robot_driver')
drive_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=1)

#create and send drive commands
move = Twist()
move.angular.z = 5
move.linear.x = 0.25
drive_pub.publish(move)
```

With this we are able to control the robot through python script! Obviously this code will be run in a loop with updates!

But how do we know where the robot is and where the path is relative to it so we can steer it? Well, we **place a camera on the robot and stream its feed to the controller**. Camera support is added through the gazebo camera sensor plugin. All we need to do is setup a subscriber to receive the camera feed and do something useful with it!

The control loop is as follows then:

1. Receive camera subscriber feed from robot
2. process image in useful data
3. use data to generate move commands
4. publish move commands to robot
5. repeat

Registering a subscriber to the camera feed is done like this:

```
image_sub = rospy.Subscriber("/robot/camera1/image_raw",Image
,self.callback,queue_size=3)
```

There is a useful librrary called CV_bride that converts this camera feed into images OpenCV can work with; we want to use OpenCV for the camera feed processing!

```
#in constructor
self.bridge = CvBridge()

#setup callback function for subscriber to call every time there is a camera
feed input
def callback(self, img):
    try:
        self.latest_img = self.bridge.imgmsg_to_cv2(img,"bgr8")
        self.empty = False
     except CvBridgeError as e:
         print(e)
```

The camera feed class is now completed!

```
class image_converter:
    def __init__(self):
        self.bridge = CvBridge()
        self.image_sub = rospy.Subscriber("/robot/camera1/image_raw",Image
,self.callback,queue_size=3)
        self.latest_img = Image()
        self.empty = True
    def callback(self, img):
        try:
            self.latest_img = self.bridge.imgmsg_to_cv2(img,"bgr8")
            self.empty = False
        except CvBridgeError as e:
            print(e)
```

Every time the robot camera sends over video frames to the subscriber, this class converts and stores the latest image!

Now we need to process this image. In lab two we worked on image processing. We will convert image colours and detect the path colour and then find the path position relative to the robot centre and call that 'displacement'

This displacement can then be scaled into useful error values in a certain range!

```
                image = self.camera.latest_img

                #image/dimension bounds
                xcenter = image.shape[1]/2
                max_reading_error = image.shape[1]/2
                min_reading_error = 25
                h_threshold = image.shape[0] - 200

                #convert colour
                gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
                image_gray = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)

                # cv2.imshow("Image window", image_gray)
                # cv2.waitKey(1)

                gray_clr = [128,128,128]

                #locate path pixels based on colour
                Y,X = np.where(np.all(image_gray==gray_clr,axis=2))
                if X.size != 0 and Y.size != 0:
                    self.move_state = 0
                    displacement = xcenter - int(np.average(X))
                    sign = displacement/np.abs(displacement)

                    # xerror = map(np.abs(displacement), min_reading_error,
                    # max_reading_error, 0, max)
                    xerror = sign*np.interp(np.abs(displacement),
                    [min_reading_error,max_reading_error],
                    [0,max_error])
```

We could simply generate move commands based on this scaled error, however this is very unstable and results in over steering.

The better approach is to use PID control (proportionality - integration - derivative). This is an important concept in control theory and is very applicable to many control situations. I will try to explain it to the best of my knowledge. Essentially the code above has given us an error value representing how far the robot is off the path. We could simply tell the robot to steer to the left if the error indicates it is to the right, but by how much?

Now consider that the control loop I mentioned previously with 5 steps has a time delay between receieving the data from the camera and outputting movement data to the robot. In a real world or multi threaded simulation, the robot is still moving while the loop is trying to calculate how much it should steer. This delay in feedback causes the robot to oversteer or understeer and we end up seeing it either oscillate wildy on the path or stray completely off the path.

Ever recall watching professional racecar driver do a quick turn of the steering wheel before the vechile even reacts? Well they know about the delay and they turn accordingly before they receive feedback.

Back to PID control : we're trying to account for the feedback delay and avoid osccillation. Proportionality represents prodocing feedback control proportional to the error receive ~ a large error should have a proportionally large steering command.

Integration accounts for past values of the error by integrating the error over time. This helps prevent reisidual error in control by taking into account historic error values.

Differentiation is taking the derivative of the error over time, usually the time step is based on the interval between each control loop. This helps predict future error trends.

If you implement steering proportional to error, you have control that's unstable. If you add the derivative term you are able to predict the error trend so as to not over steer or under steer.

The code looks like this with scaling parameters Kd and Kp for tuning.

```
        def calculate_pid(self,error):
            curr_time = rospy.get_time()
            time_step = curr_time - self.last_time
            error_step = error - self.last_error

            if time_step == 0:
                time_step = 1
            #inst. derivative
            derivative = error_step/time_step

            d=K_D*derivative
            p=K_P*error

            return d+p

        def get_move_cmd(self):
            error = self.get_error()
            move = Twist()

            if(self.move_state == 0):
                g = self.calculate_pid(error)

                move.angular.z = g * multiplier
                move.linear.x = np.interp(move.angular.z,
                        [0,max_error],
                        [0.5,0])
            elif self.move_state == 1:
                move.angular.z = 2.5
            elif self.move_state == 2:
                move.linear.x = -0.25

            print('z',move.angular.z)
            print('x',move.linear.x)

            return move
```

A common tuning method of the kp and kd parameters is to set kd to zero and increase kp until the robot begins to oscillate around the path. Then reduce kp by about half and start increasing kd to diminish oscillations (aka dont over or under steer).

Lab 3 results demo

https://drive.google.com/file/d/1LDpBjEsgrTGg_lDdnyQBowzeiiK6Yrvf/view

Lab 4 - Feb 3

GUI and tracking using sift

Skills used:

- QT and python gui creation
- SIFT tracking of realtime video feed

In [ ]: