IGV: README

This is a short explanation of the code written for the 2017-2018 Roger Williams University Intelligent Ground Vehicle, Explorator. All code files are found within the folder: “User/Navio2/Python” onboard the Raspberry Pi. The files are also available via the shared Google Drive folder and provided on the Flash Drive to Professor Baldwin.

This programming project was built around the use of the Navio2 Raspberry Pi hat for use of its GPS sensor. The Navio2 manufacturer provides custom images of Raspbian with their API built in. (https://docs.emlid.com/navio2/). Unfortunately, due to the dependencies of the provided library, all code is stored within a very specific folder location as previously noted. To create function calls for the sensors themselves, the team had to edit the provided library, this is why the code base resides here.

AccelGyroMag_edit.py

This is a variant of a provided Navio2 developer file. This file contains an IMU class which supports the Gyro, Accelerometer, and Magnometer sensor calls. Unfortunately, the Magnometer was too sensitive for use, and got a fair amount of noise from surrounding devices. An external compass module was used, explained later.

GPS_edit.py

This is a variant of a provided Navio2 developer file. This file includes one class and one function that provides the current latitude and longitude of the robot. The standard developer file GPS.py can be run to provide more information such as GPS satellite fix, etc. This function does strip the latitude and longitude to be in the specific format (##.######) for manipulation by functions within the navigation file. This is due for computational reasons.

lidar_data.py

This file opens a serial port to communicate with an Arduino Mega, which is running a continuous loop of serial outputting LIDAR data. This file will provide an error if the USB port is not chosen properly. The USB port can always be identified by running “ls dev/tty*” in terminal. The function uses a 0.8 second delay time between reading new lines as it was found without this the data parsed would often be incomplete (only getting two sensor readings).

light_trig.py

This file opens a serial port to communicate with an Arduino Uno, which simply triggers a high pin to a 555 timer when autonomous mode has begun. Once this 555 timer is turned on it will induce flashing lighting. When the lights are off, this means that they are not flashing, the green status lighting should remain on as the robot could be in teleop mode, or just generally is up and running.

line_avoidance.py

This file uses OpenCV to communicate with the USB camera and identify lines via a Hough transform. The saber_act() function was a test function used when initially testing this camera module. The main function that is being called is avg_out() which will create an average of camera readings to provide a slope to the called variable. This slope is used to determine if the detected line is too close/steep, which can be used to induce turning. In previous years, and this year if the slope were negative that would require the robot to turn left, if the slop were positive that would require the robot to turn right.

navigation.py

This navigation file/class contains all of the major solving algorithms for the robot, this includes calculations for: heading, distance; decision making for: line avoidance, and obstacle avoidance. Also includes some experimental gpsFilter() functions. The main function in this that is called for autonomous mode is gpsGO(desLat, desLon), the desired latitude and longitude for the robot to navigate to.

Sabertooth2x60.py

This file creates serial communication via a breakout USB device (CP2102) to the sabertooth2x60 motor controller. The sabertooth2x60 is using the simplified serial communcication protocal, meaning that simple character data types sent to the motor controller change which motor, and how fast it is spinning. It is very important that the CP2102 is plugged in properly to the motor controller, if it is not the robot can spin out of control.

way.py

There was a two-week period in which the robot was turning randomly and running over people. We were unsure what could be creating the issue, and as a troubleshooting step created this file, which essentially is the motor control, navigation, and sensor calls all mushed into one file. The actual issue ended up being a broken CP2102 module.

xbox.py

This file was found from an online resource and is how Xbox controller communication is established and used. This file includes a great deal of information to program different buttons on the Xbox controller, mapping of joysticks and many other useful features of Xbox controllers that I didn't even know existed.

xboxControl.py

This is the main file that is used to run the code. This file will run once started until the user presses the back button on the Xbox controller, induces a CTRL+C command to terminate the Python script, or there is some exception thrown. This file will not run if the Xbox controller is not connected, or if the compass is not connected. The Xbox controller failure is a straightforward error and explains to run the code as sudo and to try again. The compass failure will either show the BNO failure, or say something about a reset. The compass failure happens when executing the code for the first time or if you are resetting the device over and over again, this is due to the rest pin not being triggered properly.

To run this code you should open terminal and enter:
cd ~/Navio2/Python
sudo python xboxControl.py

Keep running the "sudo python xboxControl.py command until you can establish a connection. Often it is best to hold down the center button on the xbox remote while starting the system up. Please be aware the second you turn on the robot and the controller is connected that any movement in joysticks will activate the motors. The wireless e stop should be clicked and the manual e stop should be pulled to the outer position.