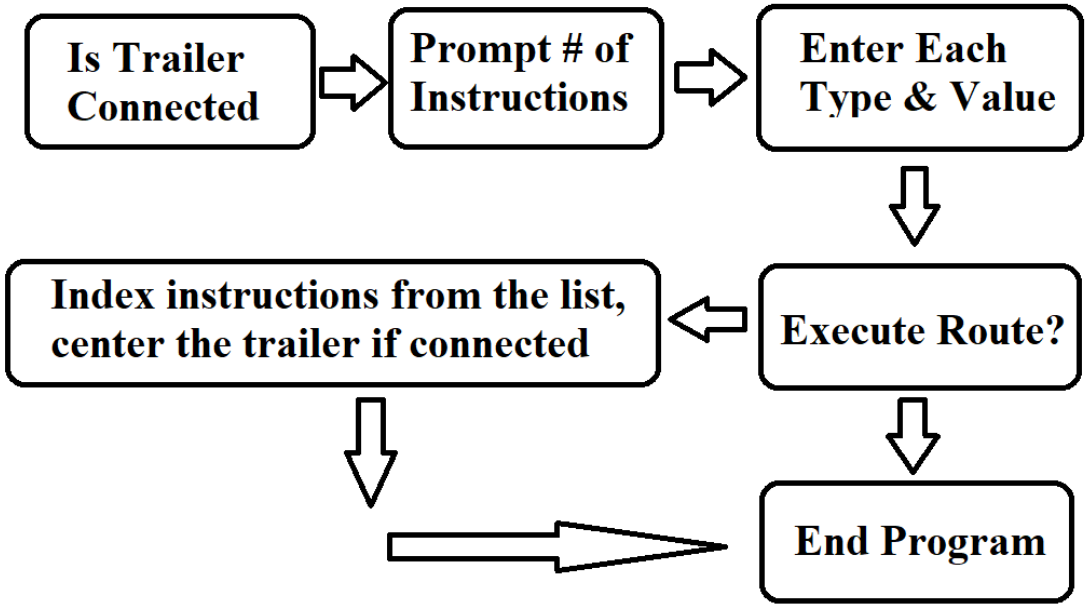
MINI Rover Pathfinding Utility

Intuitive Console-Based Route Programming

```
Does the rover have a trailer? (1 = yes, 0 = no)
0
How many instructions do you wish to input?
3
As I prompt each instruction, please specify the type
(turn, drive), then specify the desired distance or angle.

For Instruction # 0 , what is the type?

drive
What distance do you wish to travel? (cm)
23
ros2 topic pub -t 67 /cmd_vel geometry_msgs/Twist "{linear: {x: 0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}" -r 30
For Instruction # 1 , what is the type?

turn
What angle do you wish to turn to? (degrees)
23
ros2 topic pub -t 6 /cmd_vel geometry_msgs/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1}}" -r 30
Error, command not recognized. Please try again.

For Instruction # 2 , what is the type?
```

Overview

This pathfinding program allows users to specify a number of instructions, enter them chronologically, and then execute them on the rover. Its intended application is quick path creation for testing peripheral hardware and/or educational demonstrations. This program also allows for the use of our custom trailer and will add autonomous instructions to correct for changes in its hitch angle.

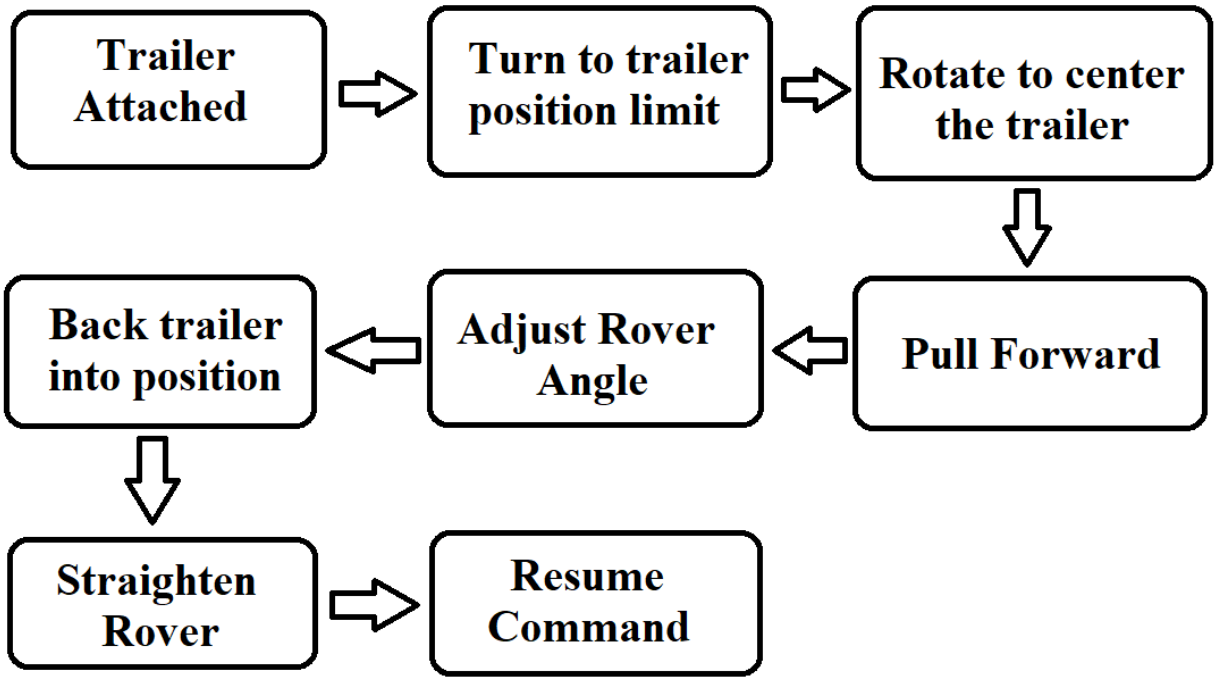


Software Overview

We first specify the GPIO location of the trailer position sensor, then run through some variables for the logic. “trailercollision” is a simple Boolean that tells us when the trailer is about to jackknife. “angleinc” represents the change in encoder units for each increment of the angle command. This value, along with the “rotationvalue” are subject to vary and should be measured manually.

```
# Program your own routes to tow or drive independently.  3-8-24

import math              # For integer rounding
import time              # For programmed delays
import os                # Used to interface w/ ROS through subshell
import RPi.GPIO as GPIO  # For GPIO communication

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO_TRAILER = 14   # Trailer collision sensor
GPIO.setup(GPIO_TRAILER, GPIO.IN)

trailercollision = 0   # 0 = about to crash, 1 = fine
anglinc = float(0.25)  # Angle mvmt per increment in encoder units
currentangl = int(0)   # Used for trailer tracking
instructiontype = []
instructionqty = []
commands = []          # List for os commands
rotationvalue = float(0.0025)  # Dist mvmt per increment in encoder units
```

The trailercenter() function is called only when the trailer is connected and the rover is either backing up or turning. Its purpose is to center the trailer before the command is executed to ensure that it does not collide with the rover during the following command. It is a simple series of arbitrary movement commands which have been tuned for proper positioning.




```python
def trailercenter():  # Trailer-centering instruction set
    sign2 = 1.0
    currentangl = int(0)  # Current angle defined for reference
    while GPIO.input(GPIO_TRAILER) == 0:  # Find trailer angle limit
        currentangle += 2  # Add two degrees to the angle
        print('ros2 topic pub -t 4 /cmd_vel geometry_msgs/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.0}}" -r 30')  # Turn 2 deg each iteration
        os.system('ros2 topic pub -t 4 /cmd_vel geometry_msgs/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.0}}" -r 30')
    print('ros2 topic pub -t 28 /cmd_vel geometry_msgs/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: -1.0}}" -r 30')  # Turn clockwise 45 deg to center tra
    os.system('ros2 topic pub -t 28 /cmd_vel geometry_msgs/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: -1.0}}" -r 30')
    currentangl -= 45  # subtract 45 degree turn from rover angle
    print('ros2 topic pub -t 92 /cmd_vel geometry_msgs/Twist "{linear: {x: 0.1, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}" -r 30')  # Drive forward 19.1"
    os.system('ros2 topic pub -t 92 /cmd_vel geometry_msgs/Twist "{linear: {x: 0.1, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}" -r 30')
    print('ros2 topic pub -t 10 /cmd_vel geometry_msgs/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: -1.0}}" -r 30')  # Turn clockwise 20 deg
    os.system('ros2 topic pub -t 10 /cmd_vel geometry_msgs/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: -1.0}}" -r 30')
    currentangl -= 20  # Add 20 deg increment to angle
    print('ros2 topic pub -t 8 /cmd_vel geometry_msgs/Twist "{linear: {x: -0.1, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}" -r 30')  # Reverse 14.1"
    os.system('ros2 topic pub -t 8 /cmd_vel geometry_msgs/Twist "{linear: {x: -0.1, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}" -r 30')
    if currentangl < 0:
        currentangl = -currentangl  # os.system cannot read negative integers for inc #
        sign2 = -1.0
    else:
        currentangl = currentangl
    anglval = int(math.ceil(currentangl * anglinc))
    print('ros2 topic pub -t %i /cmd_vel geometry_msgs/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: %i}}" -r 30' % (anglval, sign2))
    os.system('ros2 topic pub -t %i /cmd_vel geometry_msgs/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: %i}}" -r 30' % (anglval, sign2))
```

The command types are recorded using the same for-loop operation, regardless of trailer presence. It is responsible for determining three pieces of data:

1. Instruction type (drive or turn)
2. Direction of movement (forward or reverse)
3. Angle or magnitude of movement (degrees or cm)

Using these three pieces of information, the loop will create a terminal command to execute the movement, and then index it to a list of instructions. These instructions will then be later called during the execution of the program.

```
for i in range(routeinstructions):  # Collect data for each inputted instruction
    print("For Instruction #", i, ", what is the type?\n")
    intype = input()
    if intype == 'turn':  # If inputted instruction type is a turn command
        instructiontype.append('turn')
        turn = bool(1)
        sign2 = 1.0
        angle = float(input("What angle do you wish to turn to? (degrees)\n"))  # Ask for angle
        if angle < 0:  # flip angle sign if negative, compensate by changing turn direction
            angle = -angle  # os.system cannot read negative integers for inc #
            sign2 = -1.0
        else:
            angle = angle
        anglval = int(math.ceil(angle*anglinc))  # Calculate motor increments based on angle
        instructionqty.append(anglval)  # add angle value to be read by appended command
        print('ros2 topic pub -t %i /cmd_vel geometry_msgs/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: %i}}" -r 30' % (anglval, sign2))
        commands.append('ros2 topic pub -t %i /cmd_vel geometry_msgs/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: %i}}" -r 30' % (anglval, sign2))
if intype == 'drive':  # If inputted instruction type is a drive command
    sign = 0.1
    drive = bool(1)
    desiredDist = float(input("What distance do you wish to travel? (cm)\n"))*.0072435117
    if desiredDist < 0:
        instructiontype.append('driverev')
        desiredDist = -desiredDist
        sign = -0.1
    else:
        instructiontype.append('drive')
    increment = int(math.ceil(desiredDist / rotationvalue))
    instructionqty.append(increment)
    print('ros2 topic pub -t %i /cmd_vel geometry_msgs/Twist "{linear: {x: %i, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}" -r 30' % (increment, sign))
    commands.append('ros2 topic pub -t %i /cmd_vel geometry_msgs/Twist "{linear: {x: %i, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}" -r 30' % (increment, sign))
else:
    print("Error, command not recognized. Please try again.\n")
```

Lastly, the program is executed by the user. If a trailer is connected to the rover, the system will run the centering function before each turn/reverse command. If the sequence is cancelled, the program ends.

```
execute = int(input("Execute Route? (Yes = 1, No = 0)\n"))
if execute == 1:  # Execute instruction set on rover
    print(instructiontype)  # print for user reference
    print(instructionqty)
    for i in range(len(commands)):  # Run through each command in list
        print("Command Type:", instructiontype[i], "Increments:", instructionqty[i])
        if hastrailer == 1 and instructiontype[i] == 'turn':  # If trailer attached, center before turn
            trailercenter()
        if hastrailer == 1 and instructiontype[i] == 'driverev':  # If trailer attached, center before reverse
            trailercenter()
        os.system(commands[i])  # Execute command
        time.sleep(1)
else:  # Route Cancelled
    print("Route Cancelled\n")
```