Hardware

- Standard Robots
    1. Use Raspberry Pi 4s
    2. Are connected to the '**AIRLab-BigLab-5G-1**' Wi-Fi network
    3. Have username / password: pi / raspberry
    4. Have functioning motors & can move efficiently on foam tiles
    5. IPs correspond to the SUB label:
        a. SUB00 = 192.168.0.200 | SUB1 = 192.168.0.201 | SUB3 = 192.168.0.203 | SUB4 = 192.168.0.204 |
        b. Full List: [192.168.0.200, 192.168.0.201, 192.168.0.203, 192.168.0.204, 192.168.0.205, 192.168.0.206, 192.168.0.207, 192.168.0.208, 192.168.0.209]
    6. **NOTE:** SUB6 is not working
    7. Tutorial: https://github.com/Freenove/Freenove_4WD_Smart_Car_Kit_for_Raspberry_Pi/blob/master/Tutorial(ordinary_wheels).pdf
- Tank Robots:
    1. Use Raspberry Pi 3 B+
    2. Are connected to the '**AIRLab-BigLab-5G-1**' Wi-Fi network
    3. Not extensively tested
    4. Have functioning motors & can move efficiently on foam tiles
    5. Can move raise and lower the claw quickly
    6. Have trouble keeping the claw in line when opening / closing
    7. Do not have commands in either workspace to move the claw
    8. IPs: 192.168.0.54 & 192.168.0.55
    9. Tutorial: https://github.com/Freenove/Freenove_Tank_Robot_Kit_for_Raspberry_Pi/blob/main/Tutorial.pdf
- WARNING:
    1. Once you have turned the robots on you must start the Freenove server in order to send commands properly
    2. To do this download the application VNC Viewer: https://www.realvnc.com/en/connect/download/viewer/?lai_sr=0-4&lai_sl=l
    3. Once downloaded, add a new connection with the corresponding IP, enter the username / password, and follow the instructions within their tutorial to launch the server

4. This means if you want to test 3 robots you must connect to all 3 of them through VNC Viewer and manually start the server
5. **NOTE**: There is a small section within both tutorials that explains how to make the Freenove server start when the robot is powered on. Unfortunately I have had no success trying this on multiple robots

Connecting to Optitrack

- Terminal Commands
    1. ros2 launch mocap4r2_optitrack_driver optitrack2.launch.py
    2. **In a new tab:** ros2 lifecycle set /mocap4r2_optitrack_driver_node activate
    3. Keep these tabs open while you are testing

Old Code

- Customizing Parameters
    1. Everything is found in the target_pid_controller package
    2. **How are parameters passed?**
        a. In custom_robot_msgs/msg/RobotCommandMsgs.msg you can see the values needed for the robot to reach a target
        b. All values are published under the corresponding custom topic: '/robot_{i}/robot_commands'
        c. Storing and switching targets, checking if a target is reached, handling the rigid bodies, and publishing all values of RobotCommandMsgs.msg can be found in 'pid_controller_node.py'
        d. 'pid_controller_node_sub.py' contains a subscription to these custom messages and uses them to do all of the work needed
    3. **Adding robots:**
        a. In 'pid_controller_node_sub.py': add the desired IP to the self.IPs vector
        b. In 'target_pid_controller.yaml': add the Optitrack Streaming ID and a nickname to the tracked_rigid_bodies vector
            i. Example: [['499', '/SUB8'], ['502', '/SUB7']]
        c. In 'target_controller_launch.py': edit the num_agents integer to reflect the number of robots you are using
    4. **Setting targets:**
        a. Open 'target_controller_launch.py'
        b. Inside the for loop are ExecuteProcess statements that declare custom PointStamped messages

c. Using the same format, change the **x** and **y** coordinates
d. After the for loop, append the new targets to the targets vector
    i. **NOTE:** They will not necessarily be added in order, but for the most consistent results add the targets in reverse (last to first)
e. As it is setup right now, all robots will be sent to the same targets

5. **Adjusting motor speeds:**
    a. Open 'pid_controller_node_sub.py'
    b. Inside the 'translate_command' method you will find how the 'linear' and 'angular' velocity variables (calculated in 'pid_controller.py') are manipulated
    c. The robots' wheels will not move if the values passed are too small
        i. Try to keep negative values <-900 and positive values >900
    d. NOTE: If you supply too much power and the robot moves very fast, the Optitrack will not be able to update its coordinates and orientation fast enough, causing the robot to:
        i. Overshoot its targets
        ii. Turn endlessly when trying to face the next target
        iii. Drive off course and stop frequently to make adjustments

- Running the Code
    1. cd into ../**mocap4ros2_ws/Archive**
    2. Run the following commands separately:
        a. colcon build
        b. source install/setup.bash
        c. ros2 launch target_pid_controller target_controller_launch.py
    3. If 'package not found', reset the path:
        a. **AMENT_PREFIX_PATH=~/Desktop/mocap4ros2_ws/Archive/install/target_pid_controller**
    4. If the launch command is successful, the robots should begin moving within a few seconds
- Issues:
    1. If the robots do not reach their last target, they will not stop, so you must turn them off and on again and restart the Freenove server before attempting to relaunch the program

New Code

- Customizing Parameters
  1. **Everything you need to edit in this version is split into 5 separate packages:**
     a. launch_all
     b. mocap4rt_tf2
     c. robot_connector
     d. target_pid_controller
     e. target_planner

  2. **How are messages passed?**
     a. In differentialdrive_msgs/msg/DifferentialDrive.msg you can see the linear and rotational velocities that will be calculated by the PID controller
     b. In the target_pid_controller package, open 'pid_controller_node.py' in the target_pid_controller folder to see a subscription of type PointStamped to the 'target' topic
        i. This file will be actively listening to all waypoints / targets published
     c. Now go to the target_planner package and open 'target_planner_node.py' in the target_planner folder to see a publisher to the previously mentioned 'target' topic
        i. This file publishes all waypoints / targets that you set in each robot's yaml file from the launch_all package
     d. Lastly, inside the robot_connector package, 'robot_connector_node.py' will listen to the DifferentialDriveStamped topic and convert its values to robot commands
        i. These values are passed to the command_callback method and then to the translate_command method where they are adjusted to the robot's necessary thresholds and then sent

  3. **Adding robots:**
     a. In the launch_all package open 'launch_all_launch.py'
     b. Change the num_robots integer to reflect the number of robots you're testing
     c. In the same package, open the config folder and add yaml files equal to the number of robots being tested
        i. All files must follow the naming convention: 'robot_{i}.yaml'
        ii. All files must have the same formatting as the 'robot_0.yaml' file
     d. Inside each yaml, change the IP to match whichever robots you are using

e. Now go to the mocap4r2_tf2 package and open the yaml file inside the config folder
f. Using the Optitrack's Streaming ID for each robot's rigid body, append it to the tracked_rigid_bodies dictionary
    i. Ex: '{"499": "SUB 8"}, {"502": "SUB 7"}'

4. **Setting targets:**
    a. In the launch all package, open any yaml files you're using
    b. Add waypoints in the order you want the robot to travel with the format being: x, y

- Running the Code:
    1. cd into the folder
    2. Run the following commands separately:
        a. colcon build
        b. source install/setup.bash
        c. ros2 launch launch_all launch_all_launch.py
    3. There shouldn't be any errors when building or running, but the code is unfinished as of now
- Issues:
    1. The targets are recognized and the first one is published correctly, but a large majority of the time the self.target_sub subscriber in 'pid_controller_node.py' is unable to call self.set_target, so the target is never received and the self.controller.target value is None
        a. This prevents the control_loop from calling the PID controller and thus the commands are never sent to the robot
    2. On the off chance it does receive the target, however, the pose of the robot is not available and cannot be transformed
        a. I am unsure of why this happens as I never faced the issue in my old code
        b. Below is a screenshot of the error message

```
target_controller-4] [INFO] [1723137868.223275405] [target_controller]: Vehicle pose is not available.
target_controller-4] [INFO] [1723137868.322004493] [target_controller]: CONTROLLER TARGET: [-3312.  -235.]
target_controller-4] [INFO] [1723137868.322474759] [target_controller]: Could not transform world to vehicle: "world" passed to lookupTransform argument tar
et_frame does not exist.
target_controller-4] [INFO] [1723137868.322838802] [target_controller]: Vehicle pose is not available.
target_controller-4] [INFO] [1723137868.422127850] [target_controller]: CONTROLLER TARGET: [-3312.  -235.]
target_controller-4] [INFO] [1723137868.422855704] [target_controller]: Could not transform world to vehicle: "world" passed to lookupTransform argument tar
et_frame does not exist.
```