testing

March 23, 2025

0.1 Testing

0.1.1 Introduction

Testing on the project source code consists of manual testing, where the user interfaces are tested to verify their expected functions.

The tests are organised into functional modules according to the system design. The functional modules and the use cases are shown in the diagram, figure xyz. For each of the use case tests (manual) the underlying code is tested with automated unit tests.

0.2 Index of Testing

The following index sets out all of the testing in the project. The tables below index both the manual and associated unit tests. Following the index, evidence for testing is given where appropriate.

Config The following tests verify the systems configuration functions. The configuration system allows different control parameters to be arranged for different sea conditions. see section xyz for functional details

Test Number	Use Case	Summary	Type	Result
1	Default	The system behaviour when no config is present	Manual	[]
1.1		Function to correctly create default config	Unittest	
2	Create	User enters config values for a new control configuration	Manual	
2.1		Function to correctly create custom config	unittest	
3	Edit	User changes a configuration's values	Manual	[]

Test Number	Use Case	Summary	Type	Result
3.1		Function to correctly edit existing config	unittest	
4	Delete	User deletes a configuration	Manual	[]
4.1		Function to correctly delete existing config	unittest	
5	Simulate	User enables simulator mode for the selected config	Manual	[]
6	Add Plugin	Adds a sensor definition and plugin code	Manual	[]

Auto Pilot

Test Number	Use Case	Summary	Result
1	Start/Stop	User starts or stops the autopilot	[]
1.1		Function to correctly start/stop autopilot	unittest
2	Adjust (+-)	User adjusts the autopilot settings	[]
2.1		Function to correctly adjust target angle	unittest
3	Set Direction	User sets the direction for the autopilot	[]
3.1		Function to correctly set target direction	unittest

Logging

Test Number	Use Case	Summary	Result
1	Start/Stop	User starts or stops the logging	[]
2	Upload	User uploads the log data	[]
3	View	User views the log data	[]

0.2.1 Evidence

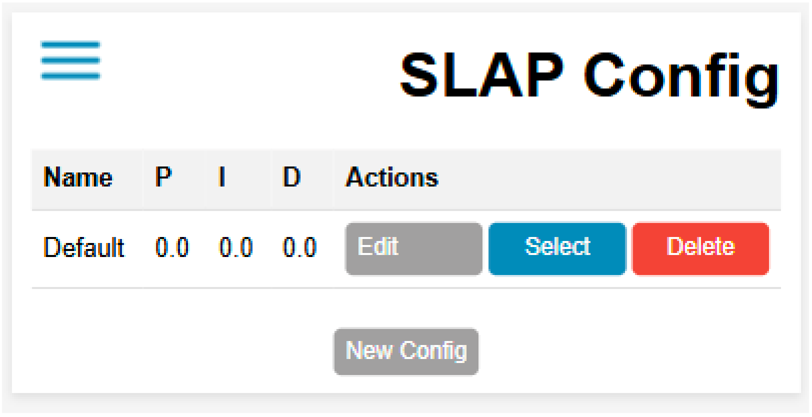
Config Test 1: Default

Description	Expected Outcome	Test Type
When the system is first started, no user config has been create and so the database will be empty. The system detects this, it creates a default config which is added to the database	Created default config	Manual

Procedure:

1. Clear database
2. Start application
3. Verify a default config has been added

Result: The screenshot shows a default config has been created



Name	P	I	D	Actions
Default	0.0	0.0	0.0	Edit Select Delete

Unit Test The unit test code below verifies the underlying methods for this functionality

```
[5]: # %load C:
  ↪\Users\franc\vscode\projects\slap\slap\src\iteration2\tests\test_defaultConfig.
  ↪py
def test_getCurrentConfig_creates_default_when_none_exists(self):
        """Test that getCurrentConfig creates a default config when none␣
  ↪exists"""
        # Get current config (should create default)
        config = self.store.getCurrentConfig()

        # Verify default config was created
        self.assertEqual(config.name, 'Default')

        # Verify config was saved to database
        self.store.cursor.execute("SELECT * FROM CONFIGS WHERE isDefault =␣
  ↪True")
```

```
        saved_config = self.store.cursor.fetchone()
        self.assertIsNotNone(saved_config)
        self.assertEqual(saved_config['name'], 'Default')
        self.assertEqual(saved_config['proportional'], 0)
        self.assertEqual(saved_config['integral'], 0)
        self.assertEqual(saved_config['differential'], 0)
```

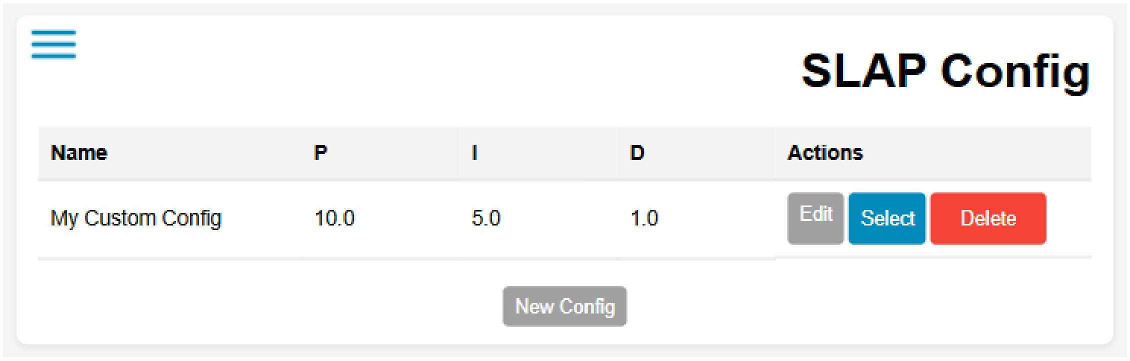
Test 2: Create

Description	Expected Outcome	Test Type
Slap provides the facility to create custom configs, the user can create a config and save it to the database to be selected for later use	Normal	Config is saved to database

Procedure:

1. Visit config page
2. Press create Config
3. Enter all needed values in the form
4. Press save
5. View saved config in database

Result: The screenshot shows a custom config has been created


SLAP Config

Name	P	I	D	Actions
My Custom Config	10.0	5.0	1.0	Edit Select Delete

Unit Test The unit test code below verifies the underlying methods for this functionality

[7]:
```
# %load C:
↪\Users\franc\vscode\projects\slap\slap\src\iteration2\tests\test_createConfig.
↪py
from services.slapStore import SlapStore, Config

def test_createConfig_creates_new_config(self):
        """Test that createConfig creates a config"""
        print("--------------------------------")
        print("")
        print("UNIT TEST: Config_Create")
        print("\nTesting creation of new config...")

        # Get current config (should create default)
        config = Config(0, "My Custom Config", 10, 5, 1)

        config = self.store.newConfig(config)

        self.assertEqual(config.name, 'My Custom Config')

        # Verify config was saved to database
        self.store.cursor.execute(f"SELECT * FROM CONFIGS WHERE configId =␣
↪'{config.configId}'")
        saved_config = self.store.cursor.fetchone()
        self.assertIsNotNone(saved_config)

        print("Verifying saved values match input values...")
        self.assertEqual(saved_config['name'], 'My Custom Config')
        self.assertEqual(saved_config['proportional'], 10)
        self.assertEqual(saved_config['integral'], 5)
        self.assertEqual(saved_config['differential'], 1)
        print("All values verified successfully")
```

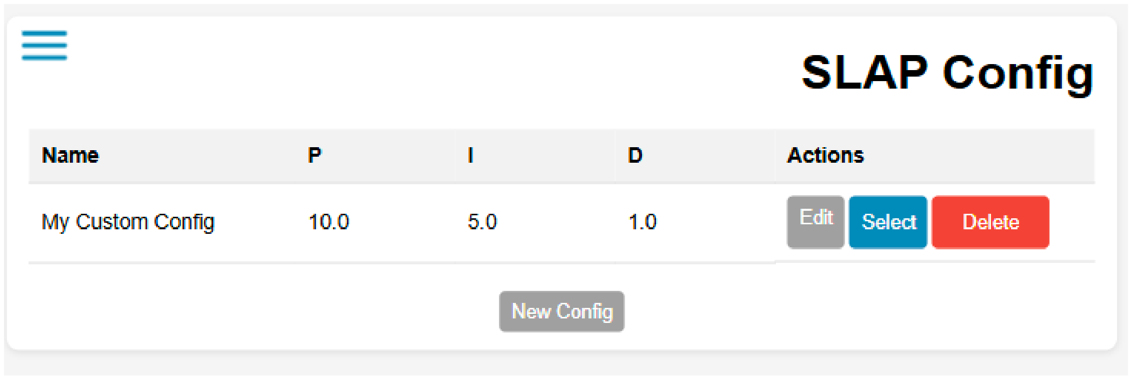
```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[7], line 2
      1 # %load C:
 ↪\Users\franc\vscode\projects\slap\slap\src\iteration2\tests\test_createConfig
 ↪py
----> 2 from services.slapStore import SlapStore, Config
      4 def test_createConfig_creates_new_config(self):
      5         """Test that createConfig creates a config"""

ModuleNotFoundError: No module named 'services'
```

Test 3: Edit

Description	Data Type	Expected Outcome	Test Type
User changes a configuration's values	Normal	Edited config is saved to database	Manual

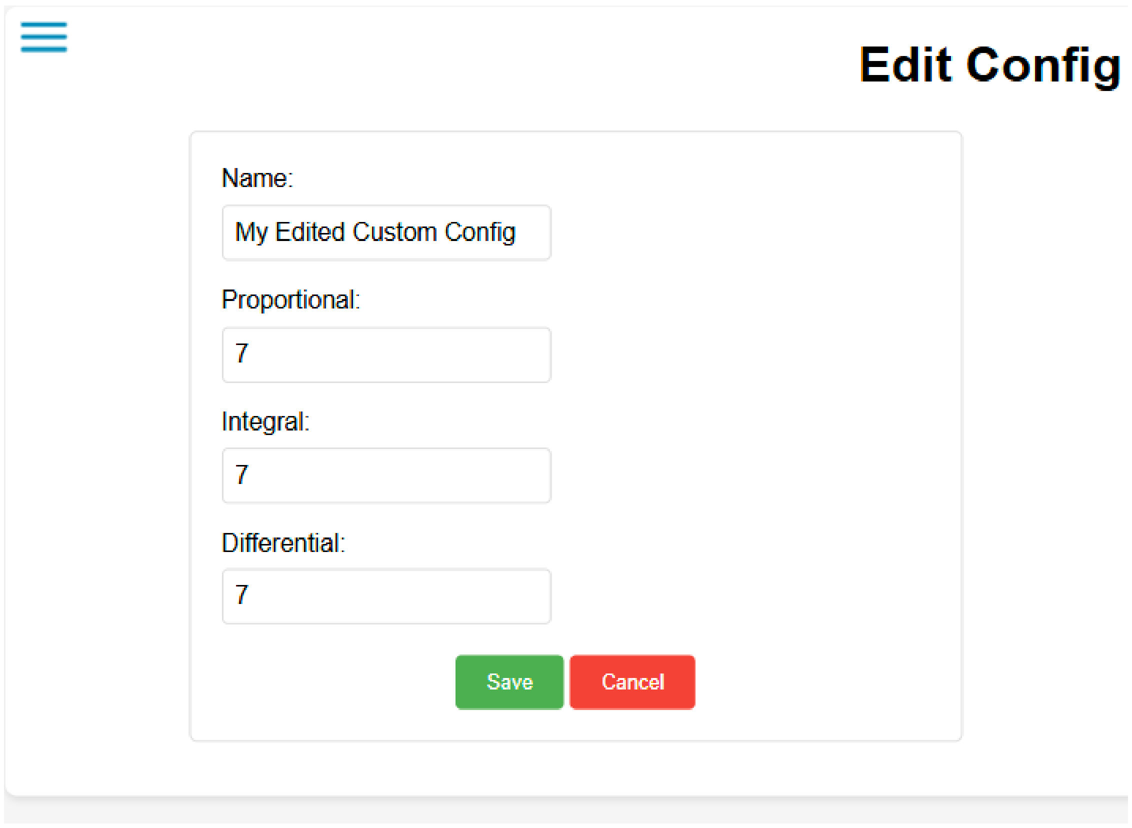
Procedure:

1. Visit config page


SLAP Config

Name	P	I	D	Actions
My Custom Config	10.0	5.0	1.0	Edit Select Delete

2. Press edit Config


Edit Config

Name: My Edited Custom Config

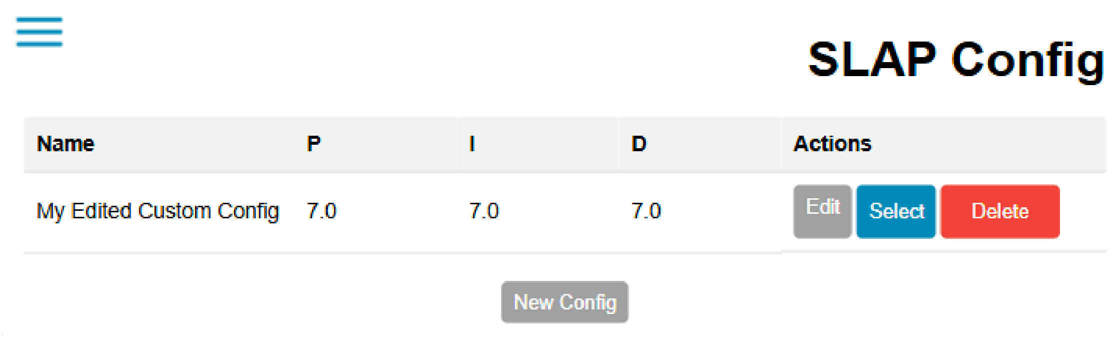
Proportional: 7

Integral: 7

Differential: 7

3. Adjust needed values in the form
4. Press save
5. View saved config in database

Result



SLAP Config

Name	P	I	D	Actions
My Edited Custom Config	7.0	7.0	7.0	Edit Select Delete

Unit Test The unit test code below verifies the underlying methods for this functionality

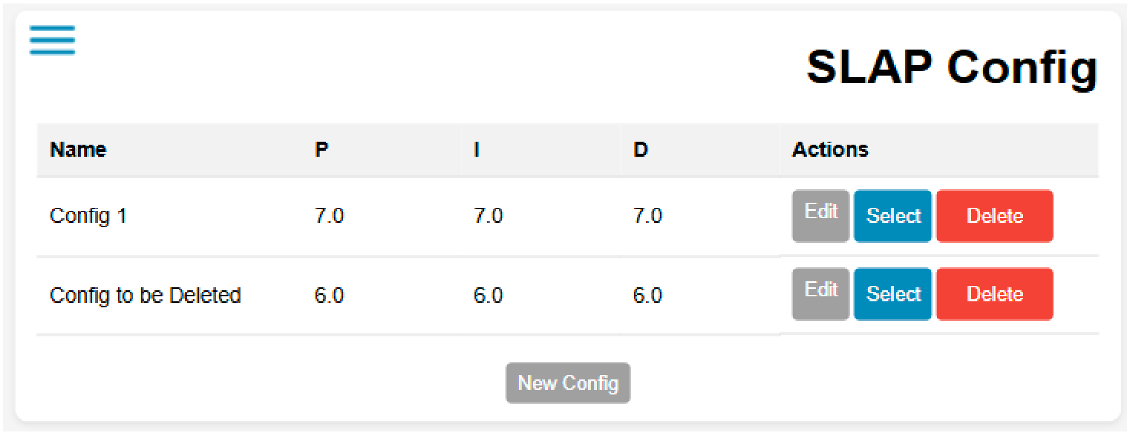
```
[ ]: # %load C:
     ↪\Users\franc\vscode\projects\slap\slap\src\iteration2\tests\test_editConfig.
     ↪py
     from services.slapStore import SlapStore, Config

     def test_editConfig_updates_existing_config(self):
             """Test that editConfig updates an existing config"""
             print("--------------------------------")
             print("")
             print("UNIT TEST: Config_Edit")
             print("\nTesting editing of existing config...")

             # Create initial config
             initial_config = Config(0, "Test Config", 1, 2, 3)
             initial_config = self.store.newConfig(initial_config)

             # Edit the config
             edited_config = Config(initial_config.configId, "Edited Config", 10,␣
     ↪20, 30)
             self.store.updateConfig(edited_config)

             # Verify config was updated in database
             self.store.cursor.execute(f"SELECT * FROM CONFIGS WHERE configId =␣
     ↪'{initial_config.configId}'")
             saved_config = self.store.cursor.fetchone()
             self.assertIsNotNone(saved_config)

             print("Verifying saved values match edited values...")
             self.assertEqual(saved_config['name'], 'Edited Config')
             self.assertEqual(saved_config['proportional'], 10)
             self.assertEqual(saved_config['integral'], 20)
             self.assertEqual(saved_config['differential'], 30)
             print("All values verified successfully")
```

Test 4: Delete

Description	Data Type	Expected Outcome	Test Type
User deletes a configuration	Normal	Config is removed from database	Manual

Procedure:

1. Visit config page

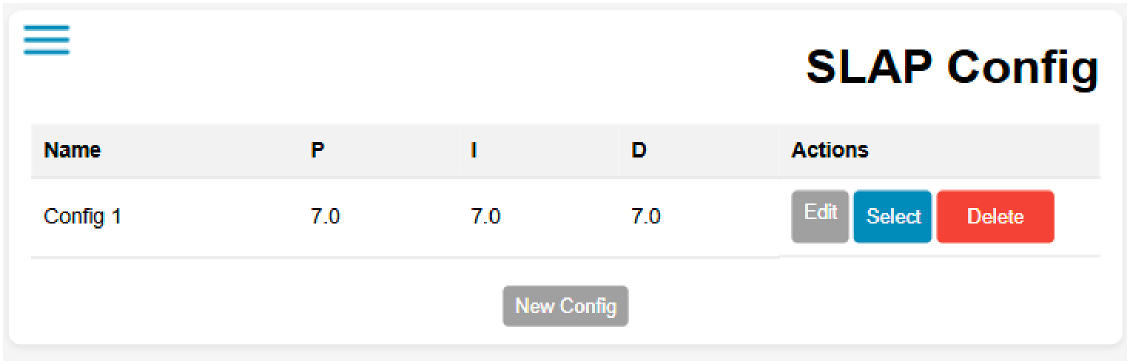


SLAP Config

Name	P	I	D	Actions
Config 1	7.0	7.0	7.0	Edit Select Delete
Config to be Deleted	6.0	6.0	6.0	Edit Select Delete

2. Press delete on a config row
3. View configs in database to verify deletion

Result



SLAP Config

Name	P	I	D	Actions
Config 1	7.0	7.0	7.0	Edit Select Delete

Unit Test The unit test code below verifies the underlying methods for this functionality

```
[ ]: # %load C:
     ↪\Users\franc\vscode\projects\slap\slap\src\iteration2\tests\test_deleteConfig.
     ↪py
     from services.slapStore import SlapStore, Config

     def test_deleteConfig_removes_existing_config(self):
             """Test that deleteConfig removes an existing config"""
             print("--------------------------------")
```

```
        print("")
        print("UNIT TEST: Config_Delete")
        print("\nTesting deletion of existing config...")

        # Create initial config
        initial_config = Config(0, "Test Config", 1, 2, 3)
        initial_config = self.store.newConfig(initial_config)

        # Delete the config
        self.store.deleteConfig(initial_config.configId)

        # Verify config was deleted from database
        self.store.cursor.execute(f"SELECT * FROM CONFIGS WHERE configId =␣
↪'{initial_config.configId}'")
        deleted_config = self.store.cursor.fetchone()

        print("Verifying config was deleted...")
        self.assertIsNone(deleted_config)
        print("Config deletion verified successfully")
```

Test 5: Simulate

Description	Data Type	Expected Outcome	Test Type
User enables simulator mode for the selected config	Normal	Simulator is started	Manual

Procedure:

1. Visit config page
2. Press simulate on a selected config row
3. Verify simulator starts with displays config name

[Screenshot of config in database]

Test 6: Add Plugin

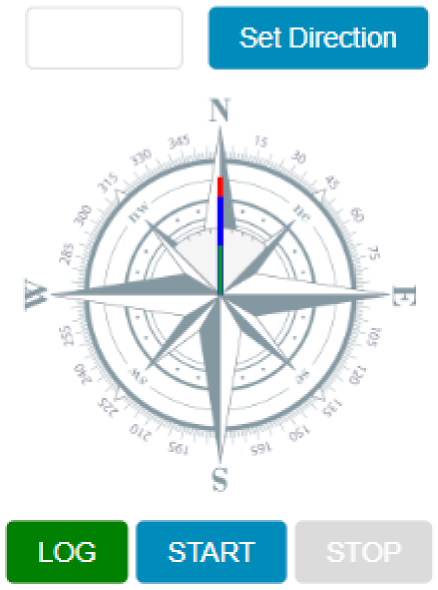
Description	Data Type	Expected Outcome	Test Type
Adds a sensor definition and plugin code	Normal	Plugin is saved to database	Manual

[Screenshot of config in database]

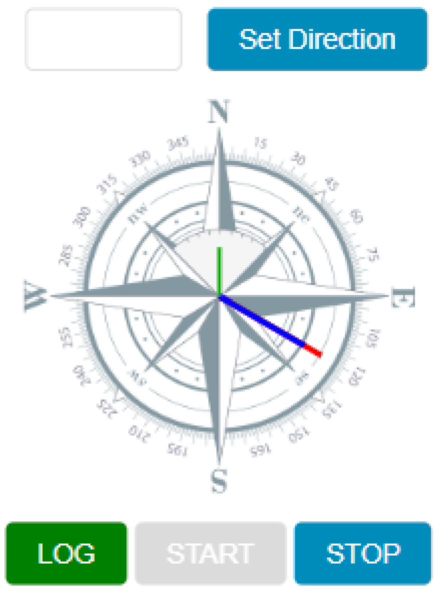
Auto Pilot   Test 1: Start/Stop

Description	Data Type	Expected Outcome	Test Type
User starts or stops the autopilot	Normal	Autopilot starts or stops	Manual

Procedure:



1. Press start/stop button
2. Verify autopilot status changes

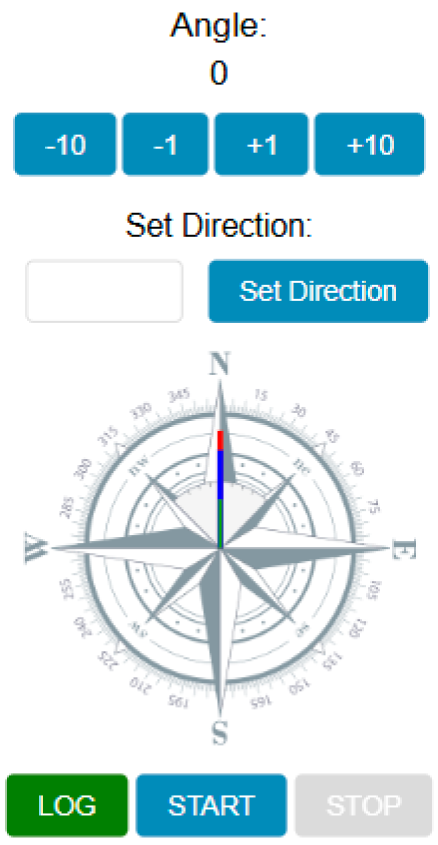


[Screenshot of autopilot in action]

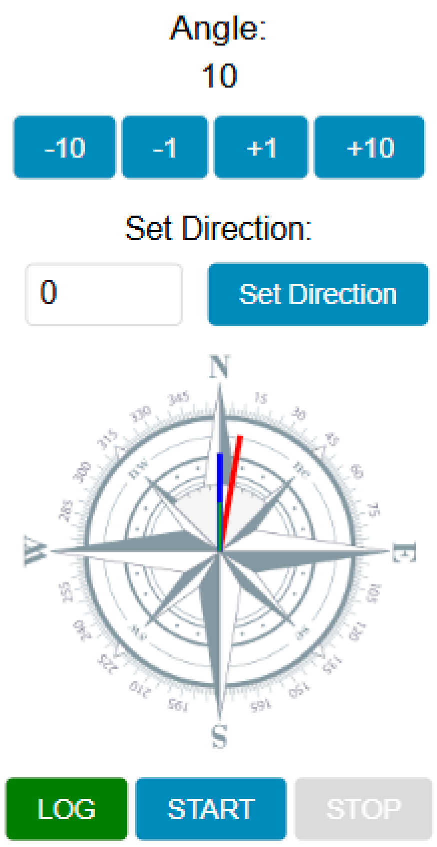
Test 2: Adjust (+-)

Description	Data Type	Expected Outcome	Test Type
User adjusts the autopilot settings	Normal	Settings are adjusted	Manual

Procedure:



2. Press +/- buttons to adjust settings
3. Verify target heading changes correctly



Unit Test The unit test code below verifies the underlying methods for this functionality

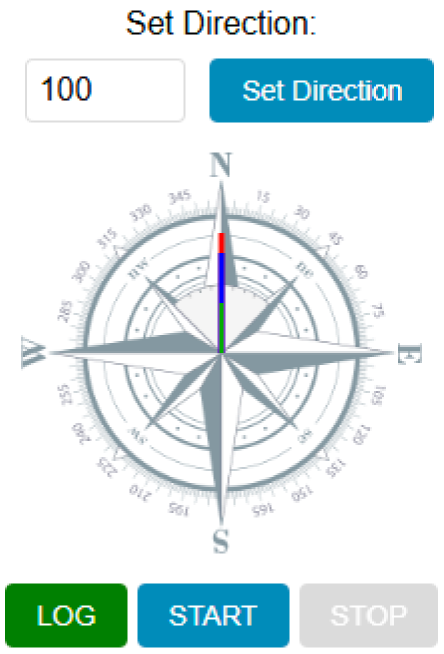
```
[ ]: # %load slap/src/iteration2/tests/test_adjustAutoPilot.py
from control.autoPilot import AutoPilot

def test_adjust_target_angle(self):
    """Test that the +/-10 buttons correctly adjust the target angle"""
    # Create test autopilot with initial target angle of 0
    auto_pilot = AutoPilot()
    auto_pilot.setHeading(0)

    # Test +10 button
    heading = auto_pilot.setHeading(auto_pilot.getHeadings()['target'] + 10)
    assert heading == 10

    # Test -10 button
    heading = auto_pilot.setHeading(auto_pilot.getHeadings()['target'] - 10)
    assert heading == 0
```

Test 3: Set Direction

Description	Data Type	Expected Outcome	Test Type
User sets the direction for the autopilot	Normal	Direction is set	Manual

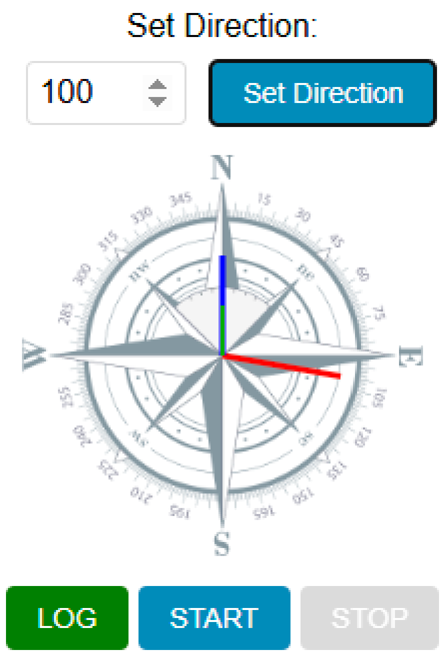
Procedure:

2. Enter desired heading in degrees (0-359)



3. Press set button
4. Verify target heading updates to entered value

Result:

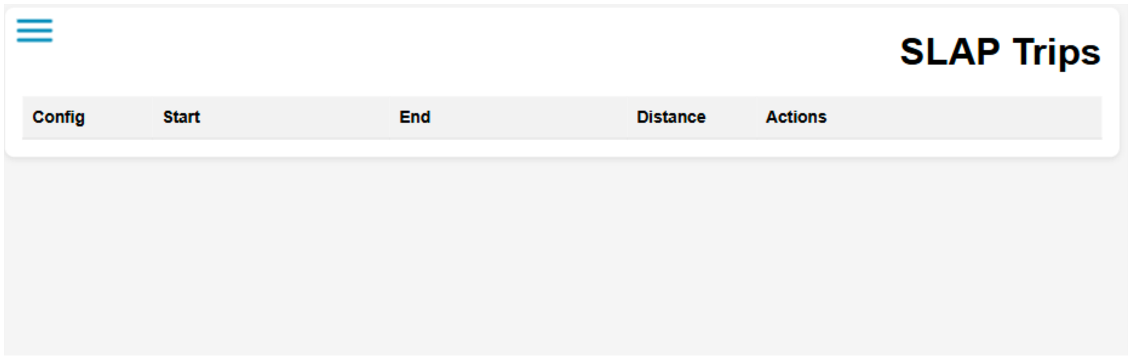


Logging Test 1: Start/Stop

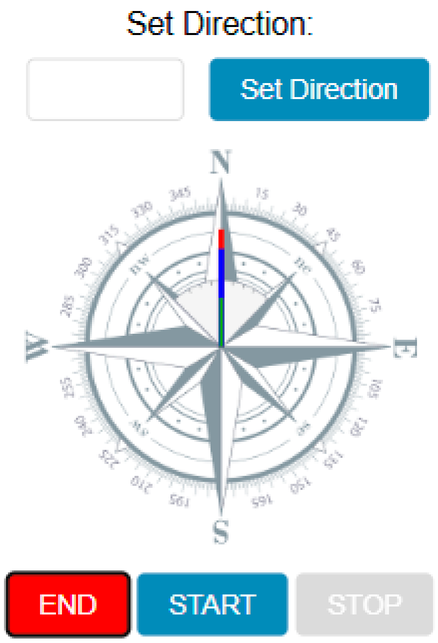
Description	Data Type	Expected Outcome	Test Type
User starts or stops the logging	Normal	Logging starts or stops	Manual

Procedure:

1. Verify no trip is present

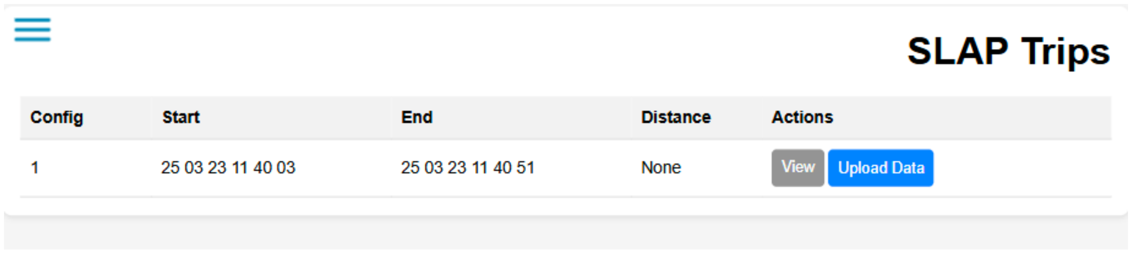


2. Press start/stop button and check UI updates



3. Check trip is created/closed appropriately

Result:



SLAP Trips

Config	Start	End	Distance	Actions
1	25 03 23 11 40 03	25 03 23 11 40 51	None	View Upload Data

Test 2: Upload

Description	Data Type	Expected Outcome	Test Type
User uploads the log data	Normal	Log data is uploaded	Manual

Procedure:

2. Press upload button
3. Verify data is sent to server
4. Verify data appears in cloud storage

[Screenshot of uploaded log data]

Test 3: View

Description	Data Type	Expected Outcome	Test Type
User views the log data	Normal	Log data is displayed	Manual

Procedure:

2. Press view button
3. Display map view to show trip

[Screenshot of log data]