NANYANG
THE INNOVATIVE POLYTECHNIC

Final Year Project Report

Development of Autonomous Drone

Student:
Ling Choon Keat

Admin No.:
171716Q

Supervisor:
Mr. Gan An Zhi

Course:
Aerospace Systems & Management

Table of Contents

Acknowledgement

I would like to express my special thanks of gratitude to my supervisor Mr. Gan An Zhi who gave me the golden opportunity to do this wonderful project, which provides me with the encouragement, support and guidance in python programming and I came to know about so many new things I am really thankful to him.

Secondly, I would also like to thank my friend, Brendon Koh Wan Jie who invited me to join this competition.

Introduction

Objective

To develop an autonomous quadcopter will be used to participate in Singapore Amazing Flying Machine Competition (SAFMC).

To develop a path planning python script for fully autonomous mission.

Background

Every year, DSO National Laboratories and Science Centre Singapore organize the Singapore Amazing Flying Machine Competition (SAFMC). The Singapore Amazing Flying Machine Competition (SAFMC) is an annual competition where local and international flying enthusiasts gather to showcase their innovative flying machine designs and put them to the test through competition against one another.

I volunteer to participate in the Singapore Amazing Flying Machine Competition Category D2 team, which required us to design and build an autonomous small air platform to perform a multitude of tasks in an indoor open course.

Scope

Develop a localization system for the autonomous quadcopter to navigate through obstacle in an indoor environment and planning a path planning script to enable fully autonomous mission.

Problem Statement

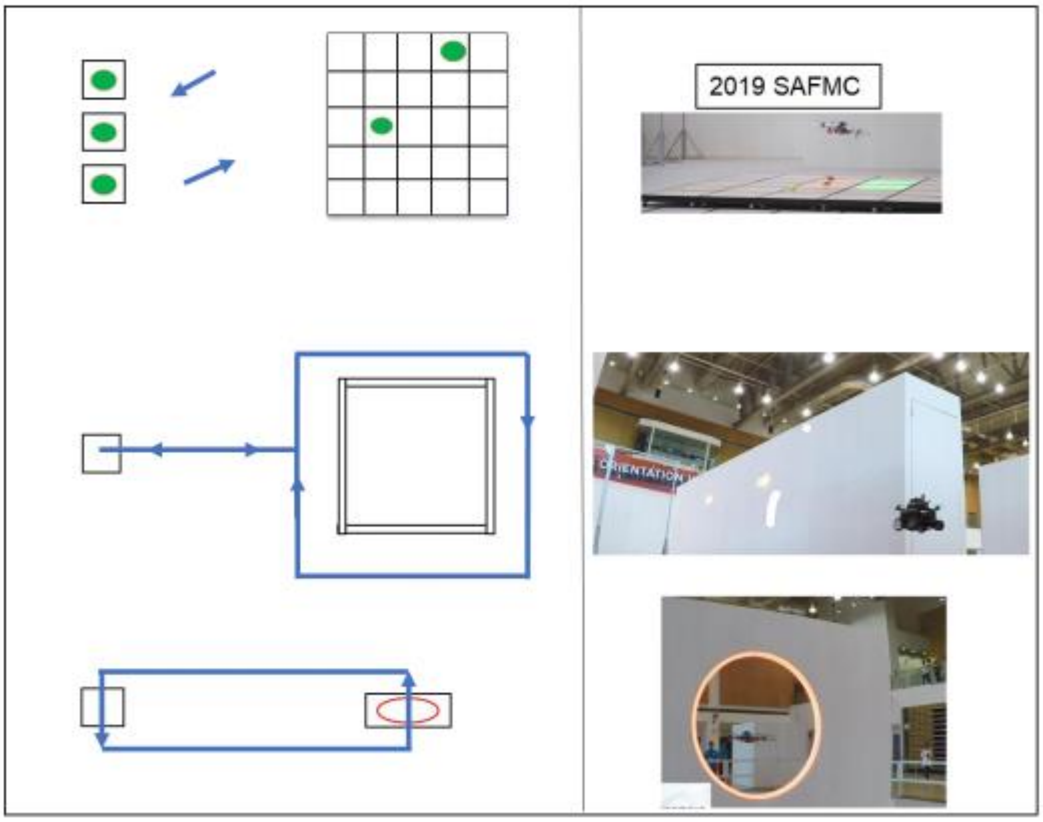
Deploy Defense Sensors: The aircraft is to deliver up to 3 payloads to the grid, ensuring that each payload lands in separate squares without touching the black borders. Bonus points are awarded if the payloads are delivered to the corner squares, which will be lit up in GREEN. Single or multiple payloads may be manually loaded, at the take-off/landing pads when the aircraft has landed. Points are awarded for landing in the landing pad if at least 1 payload is delivered successfully.

Border Surveillance: The aircraft is to patrol the border completely, covering the full perimeter of the square. The flight path will be limited to a 5-meter corridor around the square wall. Bonus points are awarded for spotting features (such as a picture or a number) that will be placed on the border walls. These features will be placed at a height of between 1.2m to 1.8m off the ground. Successful spotting of the features is determined through the live feed on the on-board camera or a saved image capture. Autonomous operation of the aircraft is expected from take-off to landing.

Window fly-through. The aircraft is to fly through the circular window. Autonomous operation of the aircraft is expected from take-off to landing. The take-off pads are 700mm by 700mm by 3mm and each light box square is 700mm by 700mm inclusive of the black borders. Any ground aids used for take-off or landing must be placed at the designated locations – points for landing will only be awarded if the aircraft lands within the designated square area (based on aircraft centroid from top view), irrespective of external ground aids employed. i.e. If a drone box larger than the landing pad is used, points will only be awarded if the drone lands in the designated landing area in the drone box.

Overview

Competition Setup



- The competition setup for Category D2 is as shown above. Teams can choose to participate in any or all the 3 missions to accrue the highest number of points within a 15-minute timeframe. Another 15 minutes will be given for setup/calibration immediately before their mission attempt. There will be a running clock, where the mission time automatically starts after the setup time.
- Each mission is a standalone and can be undertaken in any order. No penalties are applied if the missions are not attempted or failed. Teams can manually shift their aircraft to and from the respective take-off and landing pads once the aircraft is safely landed.
- If the team needs to repair/troubleshoot the aircraft, they are required (if possible) to land the aircraft either on the take-off and landing pads or on the grid. All repairs/troubleshooting should be done either on the take-off and landing pads or outside of the playing field.

Project Description

The competition arena will be held in an indoor environment; indoor positioning systems cannot make use of Global Positioning System (GPS) signals because they are too weak to penetrate indoors. Therefore, the drone cannot be able to navigate through the arena using GPS. a localization system for indoor environment need to be develop.

What is Simultaneous Localization and Mapping (SLAM)?

To get around, robots need a little help from maps, just like the rest of us. Just like humans, bots can't always rely on GPS, especially when they operate indoors. And GPS isn't sufficiently accurate enough outdoors because precision within a few inches is required to move about safely. Instead they rely on what's known as simultaneous localization and mapping, or SLAM, to discover and map their surroundings.

Using SLAM, robots build their own maps as they go. It lets them know their position by aligning the sensor data they collect with whatever sensor data they've already collected to build out a map for navigation.

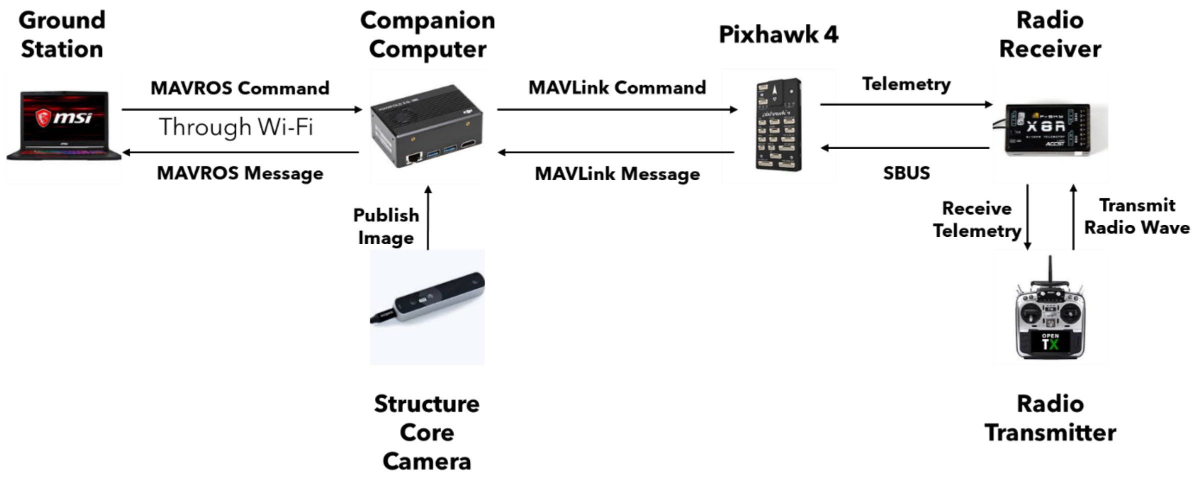
In this project, I will be using the Structure Core to develop a localization system using source code provides from online tutorial. This source code is a SLAM approach that can be used without odometry and even in platforms that exhibit roll/pitch motion (of the sensor, the platform or both). It reads the infrared image scan data from the quadcopter and returns 2D pose estimation.

Project Requirements

- Configure onboard computer (NUC) and laptop
- Develop a localization and navigation system for the autonomous drone
- Integrate the vision information (distance between the center point to the quadcopter) into the path planning script
- Writing a path planning python script for fully autonomous mission
- Create a user interface for the ease of trouble shooting and adjustment for drone

Project Development

System Overview



- Robot Operating System (ROS): ROS provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. A ROS system is comprised of several independent nodes, each of which communicates with the other nodes using a publish/subscribe messaging model.
- Primary Base Station GUI (QGroundControl): All the code in QGroundControl is open-source and it provides full flight control and mission planning for any MAVLink enabled drone.
- Secondary Base Station GUI (rqt): This base station was develop using Qt Creator (GUI) and it can be customized to our needs. It displays the important parameters to be monitored and additional feature such as flight mode selection, sending of position setpoint and activation of release mechanism.
- Router: Provide the data transfer between the laptop and the onboard computer using Wi-Fi.
- On-board Computer: The high-level control is performed by the DJI Manifold 2C as an on-board computer, using ROS as an implementation framework.

- MAVROS: Micro Aerial Vehicle Robotic Operating System (MAVROS) is a comprehensive ROS package that makes use of the Micro Air Vehicle Link (MAVLink) communication library. It serves as a gateway for sending and receiving information between ROS nodes and a MAVLink compatible Flight Controller (FC). MAVROS provides a method of connecting to a FC over serial port, User Datagram Protocol (UDP) or TCP. This flexibility is important for software testing with a simulated FC executing locally, allowing for cohesion between FC hardware execution and SITL simulation testing. Acting as a bridge from MAVLink to ROS, MAVROS supplies the medium to interface the onboard computer with the Pixhawk FC for high-level control.
- MAVLink: MAVLink is a parallel outcome of the PX4 project used for communication between the Flight Controller Unit and the ground station or the onboard computer. It is designed as a marshalling library, i.e. serializing the defined messages for transmission.
- Gazebo: The Gazebo simulation environment was used for software testing for its modeling capabilities and compatibility with ROS and PX4 Firmware, allowing for Software-In-The-Loop (SITL) testing of quadcopter, future algorithms and platform designs that closely simulate actual conditions.

Hardware Development

- External Hardware
  - Laptop: *MSI GE73VR 7RE Raider*
    - CPU: Intel(R) Core (TM) i7-7700HQ
    - GPU: GeForce® GTX 1060 with 6GB GDDR5
    - Display: 17.3" FHD (1920x1080), 120Hz, 5ms, Wide-View
    - Storage: 256GB SSD & 1TB HDD
    - RAM: 16 GB
    - Weight: 2.8 kg

- *Wireless Router: D-Link DIR-842 Wireless AC1200 Dual Band Gigabit Router*
    - Device Interfaces:
        - IEEE 802.11 ac/n/g/b/a wireless LAN
        - 10/100/1000 Gigabit Ethernet Wan port –
        - Four 10/100/1000 Gigabit Ethernet LAN ports
    - Operating Frequency:
        - 2.4 GHz band: 2400 - 2483.5 MHz
        - 5 GHz band: 5150 - 5725 MHz
    - Security:
        - WPA &WPA2 (Wi-Fi Protected Access)
        - WPS (Wi-Fi Protected Setup)

- *Radio Controller Transmitter (2.4GHz): Jumper T16 Pro*
    - Size: 180*190*58
    - Weight: 888g
    - Voltage: DC7-8.4V
    - Current: 350mah (NO CRSF)
    - Channel: 16ch
    - Internal Jumper JP4-in-1 Built in Module

- Quadcopter Hardware
    - *Quad X Carbon Fiber Frame and Soft Buffer Landing Gear*
        - Board and Component Support: carbon fiber frame, light and strong
        - Copter Arm: carbon fibre tube 20mm diameter
        - Arm Mount: tube clipper 20mm diameter
        - Size: 500mm (diagonal measurement)
        - Avionics Support: nylon spacers (light and insulated)
- Quadcopter Avionics Module
    - *Holybro Pixhawk 4*
        - Technical Specifications
            - Main FMU Processor: STM32F765
                - 32 Bit Arm ® Cortex® -M7, 216MHz, 2MB memory, 512KB RAM
            - IO Processor: STM32F100
                - 32 Bit Arm ® Cortex® -M3, 24MHz, 8KB SRAM
            - On-board sensors
                - Accel/Gyro: ICM-20689
                - Accel/Gyro: BMI055
                - Mag: IST8310
                - Barometer: MS5611

- Interfaces
  - 8-16 PWM servo outputs (8 from IO, 8 from FMU)
  - 3 dedicated PWM/Capture inputs on FMU
  - Dedicated R/C input for CPPM
  - Dedicated R/C input for Spektrum / DSM and S.Bus
  - with analog / PWM RSSI input
  - Dedicated S.Bus servo output
  - 5 general purpose serial ports
    - 2 with full flow control
    - 1with separate 1.5A current limit

  - 3 I2C ports
  - 4 SPI buses
    - 1 internal high-speed SPI sensor bus with 4 chip selects and 6 DRDYs
    - 1 internal low noise SPI bus dedicated for
    - Barometer with 2 chip selects, no DRDYs - 1 internal SPI bus dedicated for FRAM
    - Supports dedicated SPI calibration EEPROM located on sensor module - 1 external SPI buses
  - Up to 2 CAN Buses for dual CAN with serial E
    - Each CAN Bus has individual silent controls or ESC RX-MUX control
  - Analog inputs for voltage/current of 2 batteries
  - 2 additional analog inputs
- Electrical Data
  - Voltage Ratings
    - Power module output: 4.9~5.5V
    - Max input voltage: 6V
    - Max current sensing: 120A
    - USB Power Input: 4.75~5.25V
    - Servo Rail Input: 0~36V
- Mechanical Data
  - Dimensions: 44x84x12mm
  - Weight: 33.3g

➢ *Electronic Speed Controller (ESC): F55A PROⅡ•F3 6S 4IN1 32bit*

- Current: 4*55A
- Peak current (10S): 4*75A
- BEC: 10V@2.0A
- LiPo: 3-6S
- Weight: 17.5g
- Size: 45*41*7.3mm
- Mounting Hole: 30.5*30.5mm

- *Radio Controller Receiver (2.4GHz): X8R*
  - Weight: 16.6g
  - Case: 46.25 x 26.6 x 14.2mm (L x W x H)
  - Operating Range: full range (>1.5km)
  - Operating Voltage Range: 4.0~10V
  - Operating Current: 100mA@5V
  - Number of Channels: 16CH (1~8ch from conventional channel outputs, 1~16ch from SBUS port or combine two X8R to become a 16 channels receiver)

- *Lidar Lite V3*
  - Power (operating voltage): 4.75-5 VDC; 6 V Max
  - Current consumption: 105ma, idle; 130ma, continuous
  - Accuracy: +/- 2.5 cm at distances greater than 1 meter. Refer to operating manual for complete operating specifications.
  - Range: 5 cm to 40 meters
  - Update rate: up to 500 Hz

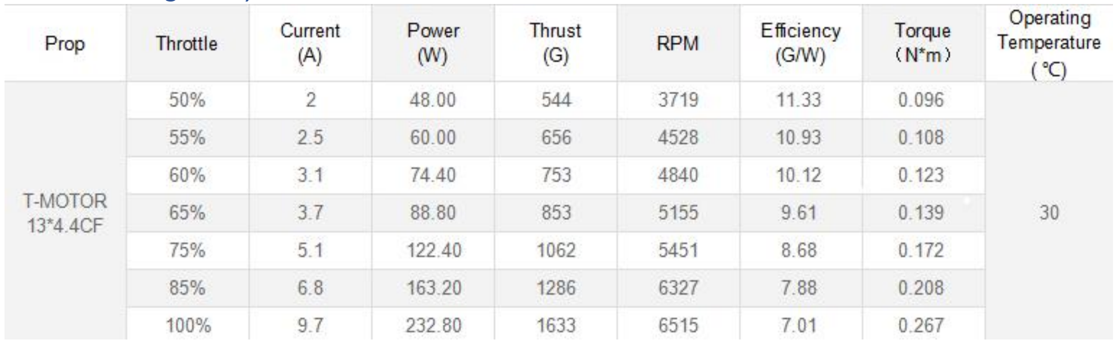
- *Localization Sensor: Structure Core (Color)*
  - Dimensions: 109*24*18mm
  - Weight: 52.5 grams
  - Depth Processing: On-device using NU3000 ASIC
  - Depth Min/Max Z Distance: 0.3-5m. Up to 10m (Depending on scene & lighting conditions.)
  - Depth Precision: ± 0.29% (Plane-fit RMS at 1m.)
  - Depth Resolution & Frame Rate: 1280 x 960 @ 54 FPS
  - Depth FOV (H x V x D): 59° x 46° x 70°
  - Visible Resolution & Frame Rate: 640 x 480 @ 100 FPS
  - Visible FOV (Diagonal): 85°
  - IMU
    - Bosch BMI055
    - 6-Axis (Gyro and Accelerometer), up to 1000 Hz
  - Up to 2000Hz if streaming only Gyro or Accelerometer
  - Projector: Laser projector module (infrared, Class 1 rating for eye safety)

  - Use Environment: Indoor / Outdoor
  - Power:2.0W (typical active), 3.1W (typical maximum)
  - Connectivity: USB 3 (Type-C), USB 2.0 operation supported
  - Operating Temperature (Ambient):0° to 35° C (32° to 95° F) (ambient, open air)

- *Companion Computer: DJI Manifold 2C*
  - Weight: Approx. 205 g
  - Dimensions: 91 x 61 x 35 mm
  - Processor: Intel Core i7-8550U
  - Memory: 8GB
  - Storage: 256 GB SSD
  - Network
  - 1000Mbps Ethernet RJ-45 Port
  - USB: USB 3.0 Port (Type A) × 2, USB 3.0 Port (Micro-B) × 1
  - I/O: UART Port 1
  - Power: 5 - 60 W

- Quadcopter Power Module
  - *6S LiPo Battery: Zolta 35C 6S1P 5200mAh*
    - Capacity: 5200mAh
    - Voltage: 6S1P / 6 Cell / 22.2V
    - Weight: 708g
    - Discharge Connector: XT90
    - Balance Connector: JST-XH (with plug saver)
    - Continuous Discharge: 35C
    - Burst Discharge: 70C

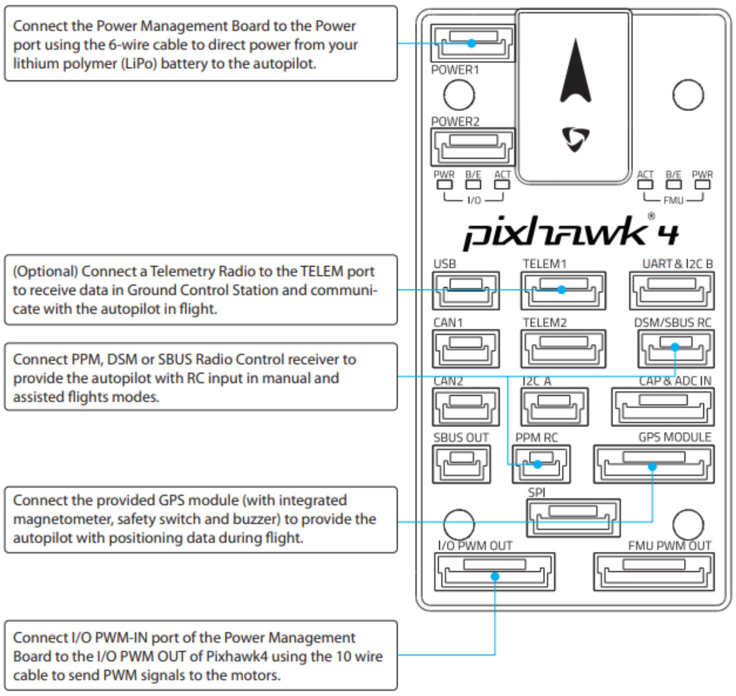
- Quadcopter Thrust Module
  - *Motor: Antigravity 4006 KV380*



Prop	Throttle	Current (A)	Power (W)	Thrust (G)	RPM	Efficiency (G/W)	Torque (N*m)	Operating Temperature (℃)
	50%	2	48.00	544	3719	11.33	0.096	
	55%	2.5	60.00	656	4528	10.93	0.108	
	60%	3.1	74.40	753	4840	10.12	0.123	
T-MOTOR 13*4.4CF	65%	3.7	88.80	853	5155	9.61	0.139	30
	75%	5.1	122.40	1062	5451	8.68	0.172	
	85%	6.8	163.20	1286	6327	7.88	0.208	
	100%	9.7	232.80	1633	6515	7.01	0.267	

  - *Propeller: APC Multi-Rotor Propeller 13x5.5 MR*
    - Pitch (inches): 5.5
    - Propeller Diameter (in.): 13
    - Hub Diameter: 0.65 in.
    - Shaft Diameter: ¼ in.
    - Product Weight: 0.85 oz.
    - Color: Grey

- Connection
  - ➢ *Pixhawk 4 Wiring Diagram*



Connect the Power Management Board to the Power port using the 6-wire cable to direct power from your lithium polymer (LiPo) battery to the autopilot.

(Optional) Connect a Telemetry Radio to the TELEM port to receive data in Ground Control Station and communicate with the autopilot in flight.

Connect PPM, DSM or SBUS Radio Control receiver to provide the autopilot with RC input in manual and assisted flights modes.

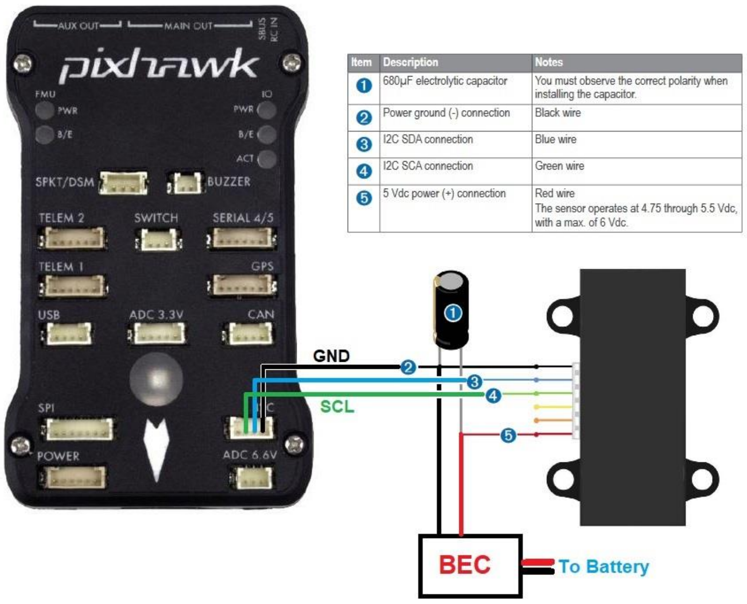
Connect the provided GPS module (with integrated magnetometer, safety switch and buzzer) to provide the autopilot with positioning data during flight.

Connect I/O PWM-IN port of the Power Management Board to the I/O PWM OUT of Pixhawk4 using the 10 wire cable to send PWM signals to the motors.

  - ➢ *FTDI Chip USB-to-serial adapter board connection*

TELEM2		FTDI	
1	+5V (red)		DO NOT CONNECT!
2	Tx (out)	5	FTDI RX (yellow) (in)
3	Rx (in)	4	FTDI TX (orange) (out)
4	CTS (in)	6	FTDI RTS (green) (out)
5	RTS (out)	2	FTDI CTS (brown) (in)
6	GND	1	FTDI GND (black)

  - ➢ *Lidar Lite V3 connection*



Item	Description	Notes
1	680µF electrolytic capacitor	You must observe the correct polarity when installing the capacitor.
2	Power ground (-) connection	Black wire
3	I2C SDA connection	Blue wire
4	I2C SCA connection	Green wire
5	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.

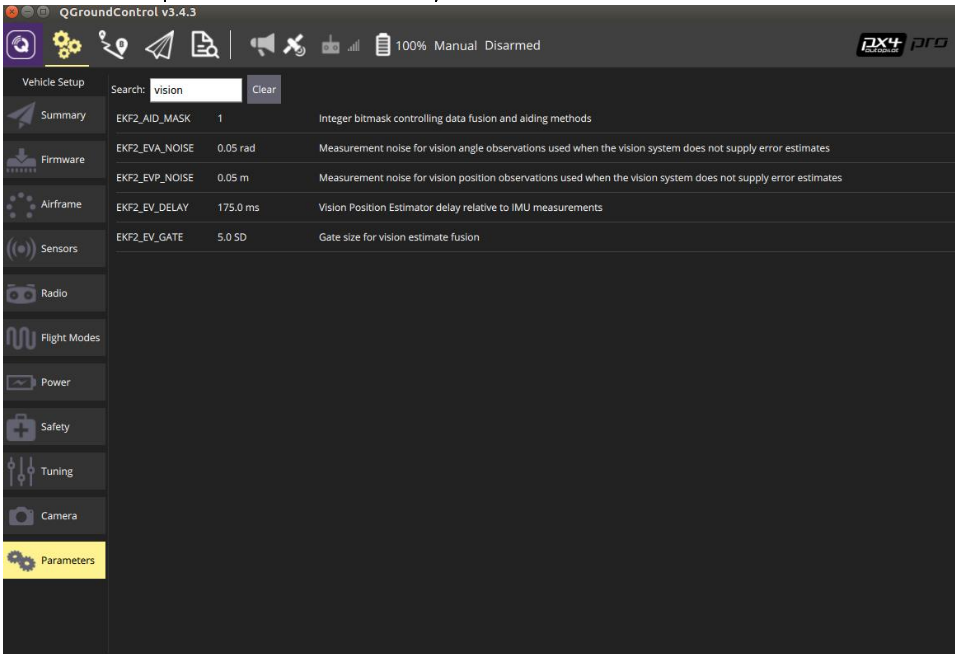
Software Development

- Configuring the laptop
  - *Install Ubuntu 16.04*
    - Follow the guide from https://www.linuxtechi.com/how-to-dual-boot-windows-10-ubuntu-16-04/
  - *Setup Gazebo testing environment by installing ROS, MAVROS, and other dependencies.*
    - For offboard control
      - Follow the guide from https://gaas.gitbook.io/guide/software-realization-build-your-own-autonomous-drone/build-your-own-autonomous-drone-e01-offboard-control-and-gazebo-simulation
    - For SLAM
    - Follow the guide from https://gaas.gitbook.io/guide/software-realization-build-your-own-autonomous-drone/build-your-own-autonomous-drone-part-3-using-slam-in-gps-denied-environment-for-position-estimation
  - *Install Qt Creator*
    - Download and install from https://www.qt.io/offline-installers
  - *Create rqt UI package file*
    - Follow the instruction form https://github.com/how-chen/rqt_mypkg
  - *Install Structure SDK package*
    - Sign up from https://developer.structure.io/sdk
    - Download the Structure SDK (Cross-Platform) zip file and follow the installation instruction inside the zip file.
- Configuring the Companion Computer
  - *Install ROS, MAVROS, and other dependencies.*
    - For offboard control
      - Follow the guide from https://gaas.gitbook.io/guide/software-realization-build-your-own-autonomous-drone/build-your-own-autonomous-drone-e01-offboard-control-and-gazebo-simulation
    - For SLAM
      - Follow the guide from https://gaas.gitbook.io/guide/software-realization-build-your-own-autonomous-drone/build-your-own-autonomous-drone-part-3-using-slam-in-gps-denied-environment-for-position-estimation

- *Install Structure SDK package*
  - Sign up from https://developer.structure.io/sdk
  - Download the Structure SDK (Cross-Platform) zip file and follow the installation instruction inside the zip file.

Configuring the Quadcopter

- *Basic configuration*
  - https://docs.px4.io/v1.9.0/en/config/index.html
- *Configuration for linking companion computer*
  - Follow the Pixhawk Setup instruction from https://dev.px4.io/v1.9.0/en/companion_computer/pixhawk_companion.html#pixhawk-setup
- *Configuration for SLAM*
  - To enable vision as the source of external position estimation, select the top left GEAR button and select "Parameters", input "vision" in the search and tick "vision position fusion" only. Click "save" to save the modification.



Search: vision

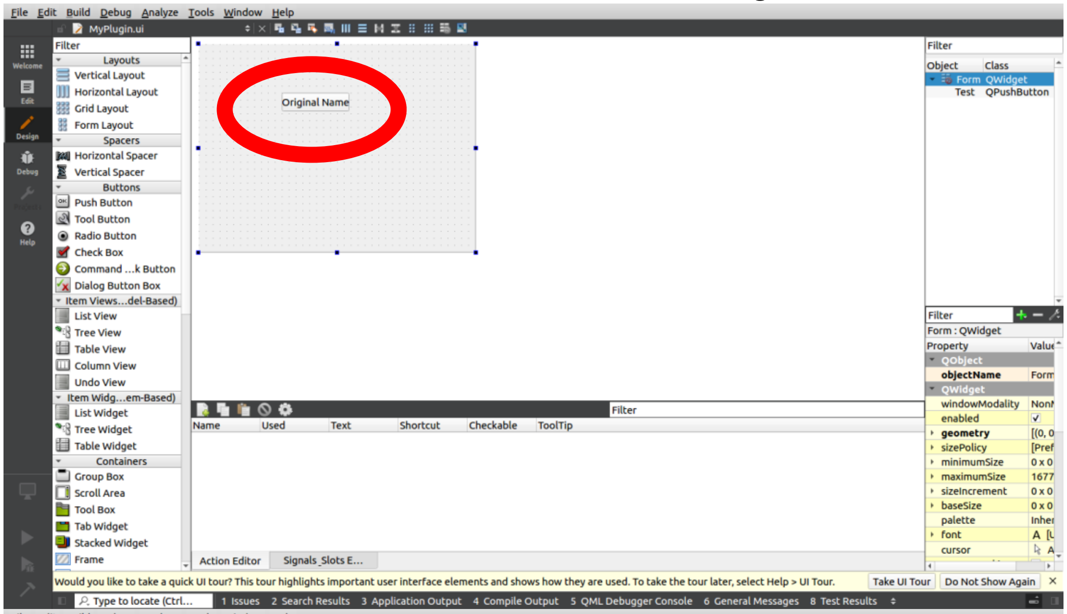
EKF2_AID_MASK	1	Integer bitmask controlling data fusion and aiding methods
EKF2_EVA_NOISE	0.05 rad	Measurement noise for vision angle observations used when the vision system does not supply error estimates
EKF2_EVP_NOISE	0.05 m	Measurement noise for vision position observations used when the vision system does not supply error estimates
EKF2_EV_DELAY	175.0 ms	Vision Position Estimator delay relative to IMU measurements
EKF2_EV_GATE	5.0 SD	Gate size for vision estimate fusion

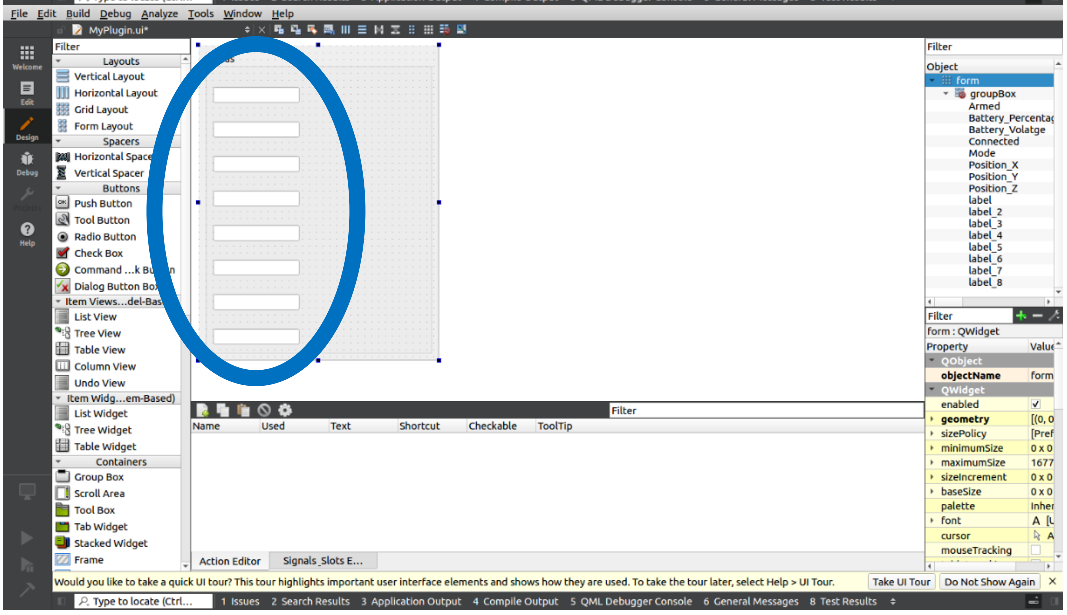
Modification for rqt UI

- *Change the UI layout*

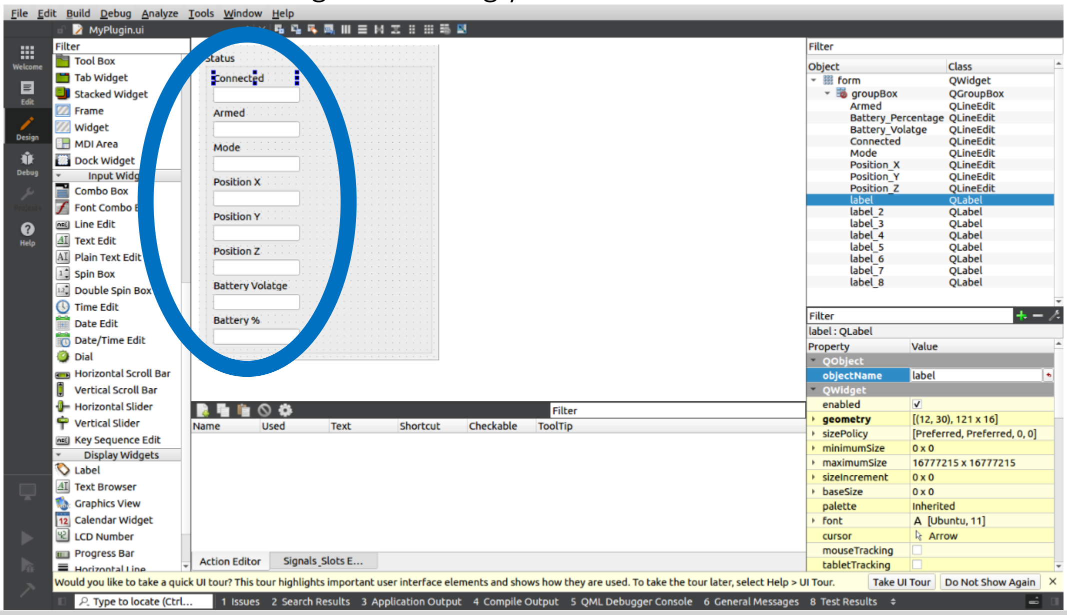
1. Open the rqt_mypkg folder under the specific catkin workspace, under the resource folder open the MyPlugin.ui with Qt Creator

2. Delete the button & add a few Label and Line Edit widget to the UI

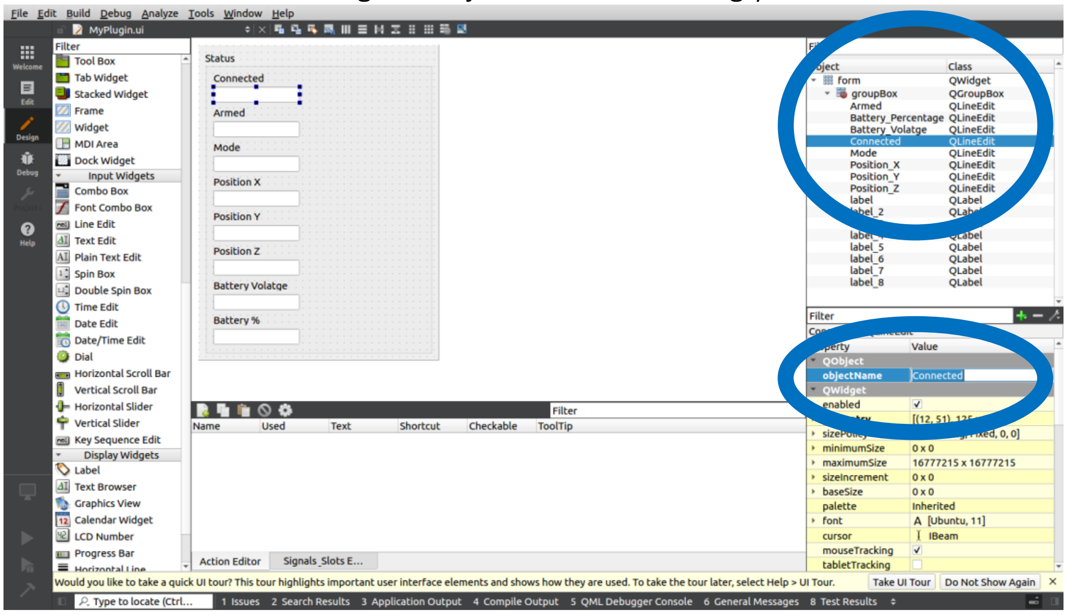




3. Rename the Label widget accordingly



4. Rename the Line Edit widget's objectName accordingly



Would you like to take a quick UI tour? This tour highlights important user interface elements and shows how they are used. To take the tour later, select Help > UI Tour.

Object	Class
form	QWidget
groupBox	QGroupBox
Armed	QLineEdit
Battery_Percentage	QLineEdit
Battery_Volatge	QLineEdit
Connected	QLineEdit
Mode	QLineEdit
Position_X	QLineEdit
Position_Y	QLineEdit
Position_Z	QLineEdit
label	QLabel

Property	Value
objectName	Connected
enabled	✓
minimumSize	0 x 0
maximumSize	16777215 x 16777215
sizeIncrement	0 x 0
baseSize	0 x 0
palette	Inherited
font	A [Ubuntu, 11]
cursor	IBeam
mouseTracking	✓

5. Save the file

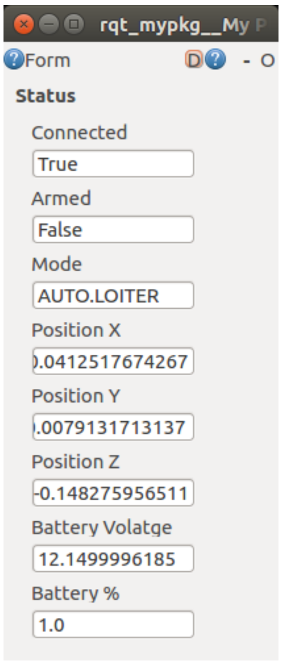
➢ *Edit the python code*

6. Open the rqt_mypkg folder under the specific catkin workspace, under the src folder open the my_module.py

7. Replace the code with this, those highlighted code are newly added to subscribe the MAVROS topic to display on the UI

```python
import os
import rospkg
import rospy

from qt_gui.plugin import Plugin
from python_qt_binding import loadUi
from python_qt_binding.QtWidgets import QWidget
# Import MAVROS message
from mavros_msgs.msg import State
from geometry_msgs.msg import PoseStamped, Point, Pose
from sensor_msgs.msg import BatteryState

class MyPlugin(Plugin):

    def __init__(self, context):
        super(MyPlugin, self).__init__(context)
        # Give QObjects reasonable names
        self.setObjectName('MyPlugin')
        rp = rospkg.RosPack()

        # Process standalone plugin command-line arguments
        from argparse import ArgumentParser
        parser = ArgumentParser()
        # Add argument(s) to the parser.
        parser.add_argument("-q", "--quiet", action="store_true",
                      dest="quiet",
                      help="Put plugin in silent mode")
        args, unknowns = parser.parse_known_args(context.argv())
        if not args.quiet:
            print 'arguments: ', args
            print 'unknowns: ', unknowns

        # Create QWidget
        self._widget = QWidget()
        # Get path to UI file which is a sibling of this file
        # in this example the .ui and .py file are in the same folder
        ui_file = os.path.join(rp.get_path('rqt_mypkg'), 'resource', 'MyPlugin.ui')

        # Extend the widget with all attributes and children from UI file
        loadUi(ui_file, self._widget)
        # Give QObjects reasonable names
        self._widget.setObjectName('MyPluginUi')
        # Show _widget.windowTitle on left-top of each plugin (when
        # it's set in _widget). This is useful when you open multiple
        # plugins at once. Also if you open multiple instances of your
        # plugin at once, these lines add number to make it easy to
        # tell from pane to pane.
        if context.serial_number() > 1:
            self._widget.setWindowTitle(self._widget.windowTitle() + (' (%d)' % context.serial_number()))
        # Add widget to the user interface
        context.add_widget(self._widget)

    # Widget
    self._widget.Connected.setReadOnly(True)
    self._widget.Armed.setReadOnly(True)
    self._widget.Mode.setReadOnly(True)
    self._widget.Position_X.setReadOnly(True)
    self._widget.Position_Y.setReadOnly(True)
    self._widget.Position_Z.setReadOnly(True)
    self._widget.Battery_Volatge.setReadOnly(True)
    self._widget.Battery_Percentage.setReadOnly(True)

    # Subscriber
    rospy.Subscriber("/mavros/state", State, self.state_callback)
    rospy.Subscriber("/mavros/local_position/pose/", PoseStamped, self.pose_callback)
    rospy.Subscriber("/mavros/battery", BatteryState, self.batt_callback)

    # Callback
    def state_callback(self, data):
    Connected=str(data.connected)
        self._widget.Connected.setText(Connected)
    Armed=str(data.armed)
        self._widget.Armed.setText(Armed)
    Mode=str(data.mode)
        self._widget.Mode.setText(Mode)

    def pose_callback(self, data):
    Position_X=str(data.pose.position.x)
        self._widget.Position_X.setText(Position_X)
    Position_Y=str(data.pose.position.y)
        self._widget.Position_Y.setText(Position_Y)
    Position_Z=str(data.pose.position.z)
        self._widget.Position_Z.setText(Position_Z)


    def batt_callback(self, data):
    Battery_Volatge=str(data.voltage)
        self._widget.Battery_Volatge.setText(Battery_Volatge)
    Battery_Percentage=str(data.percentage)
        self._widget.Battery_Percentage.setText(Battery_Percentage)


    def shutdown_plugin(self):
        # TODO unregister all publishers here
        pass

    def save_settings(self, plugin_settings, instance_settings):
        # TODO save intrinsic configuration, usually using:
        # instance_settings.set_value(k, v)
        pass

        def restore_settings(self, plugin_settings, instance_settings):
            # TODO restore intrinsic configuration, usually using:
            # v = instance_settings.value(k)
            pass

        #def trigger_configuration(self):
            # Comment in to signal that the plugin has a way to configure
            # This will enable a setting button (gear icon) in each dock widget title bar
            # Usually used to open a modal configuration dialog
```

8. Save the file
9. If you run gazebo simulation and run this package by typing *rosrun rqt_mypkg rqt_mypkg*, the MAVROS message will be show on the UI



Status

Connected
True
Armed
False
Mode
AUTO.LOITER
Position X
.0412517674267
Position Y
.0079131713137
Position Z
-0.148275956511
Battery Volatge
12.1499996185
Battery %
1.0

References

- Previous final project year report by Mr. Lim Zhi Xuan
- https://safmc.com.sg/wp-content/uploads/2019/11/SAFMC-2020-CAT-D-CHALLENGE-BOOKLET_V1.0-1.pdf
- https://blogs.nvidia.com/blog/2019/07/25/what-is-simultaneous-localization-and-mapping-nvidia-jetson-isaac-sdk/
- https://www.msi.com/Laptop/GE73VR-7RE-Raider/Specification
- https://www.team-blacksheep.com/products/prod:jumper_t16_pro
- https://shop.holybro.com/pixhawk-4_p1089.html
- http://store-en.tmotor.com/goods.php?id=915
- https://www.frsky-rc.com/product/x8r/
- https://structure.io/structure-core/specs
- https://www.dji.com/manifold-2/specs
- https://rc.zenmtech.com/lipo-zolta-35c-6s1p-5200mah-1.html
- https://dev.px4.io/v1.9.0/en/companion_computer/pixhawk_companion.html
- https://www.linuxtechi.com/how-to-dual-boot-windows-10-ubuntu-16-04/
- https://gaas.gitbook.io/guide/
- https://dev.px4.io/master/en/index.html
- http://docs.px4.io/master/en/index.html
- https://github.com/how-chen/rqt_mypkg