

# Kingfisher Manual

## Contents

Software .....	2
Shell scripts .....	2
Navigation .....	3
Driver .....	4
Monitor .....	4
Hardware .....	5
MCU .....	5
Atom PC .....	6
Others .....	6

## Software

Navigation program enables Kingfisher to follow waypoints. Navigation core module can be reused in non-ROS environment. Associated supports written for the navigation can also be used in manual control and other scenarios. Support packages for Kingfisher were not used as they become obsolete and incompatible with Ubuntu 16. ROS serial is used for communicating between Atom PC and MCU for setting motor command and receive telemetry.

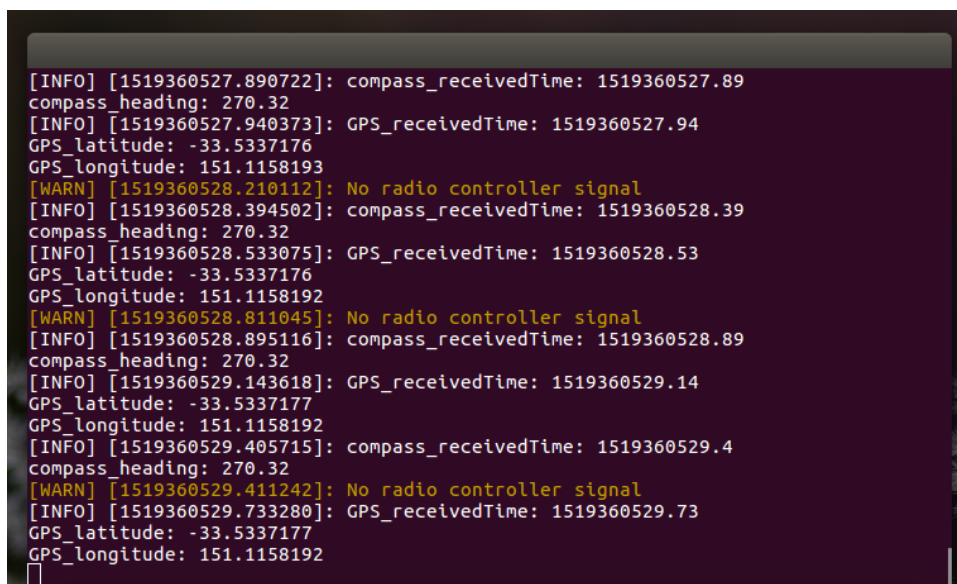
Configuration is done via config.py where parameters and waypoints coordinate for navigation are stored.

### Shell scripts

To reduce the complexity in starting up all required services in Kingfisher, a couple simple shell scripts are written. Shell scripts are available in the navigation/shellScripts/ folder for bringing up essential services including ROS core, GPS, compass, serial communication and system monitor.

```
cd shellScripts/  
./start.sh
```

which calls all other scripts and display system information and warnings as below:



```
[INFO] [1519360527.890722]: compass_receivedTime: 1519360527.89  
compass_heading: 270.32  
[INFO] [1519360527.940373]: GPS_receivedTime: 1519360527.94  
GPS_latitude: -33.5337176  
GPS_longitude: 151.1158193  
[WARN] [1519360528.210112]: No radio controller signal  
[INFO] [1519360528.394502]: compass_receivedTime: 1519360528.39  
compass_heading: 270.32  
[INFO] [1519360528.533075]: GPS_receivedTime: 1519360528.53  
GPS_latitude: -33.5337176  
GPS_longitude: 151.1158192  
[WARN] [1519360528.811045]: No radio controller signal  
[INFO] [1519360528.895116]: compass_receivedTime: 1519360528.89  
compass_heading: 270.32  
[INFO] [1519360529.143618]: GPS_receivedTime: 1519360529.14  
GPS_latitude: -33.5337177  
GPS_longitude: 151.1158192  
[INFO] [1519360529.405715]: compass_receivedTime: 1519360529.4  
compass_heading: 270.32  
[WARN] [1519360529.411242]: No radio controller signal  
[INFO] [1519360529.733280]: GPS_receivedTime: 1519360529.73  
GPS_latitude: -33.5337177  
GPS_longitude: 151.1158192
```

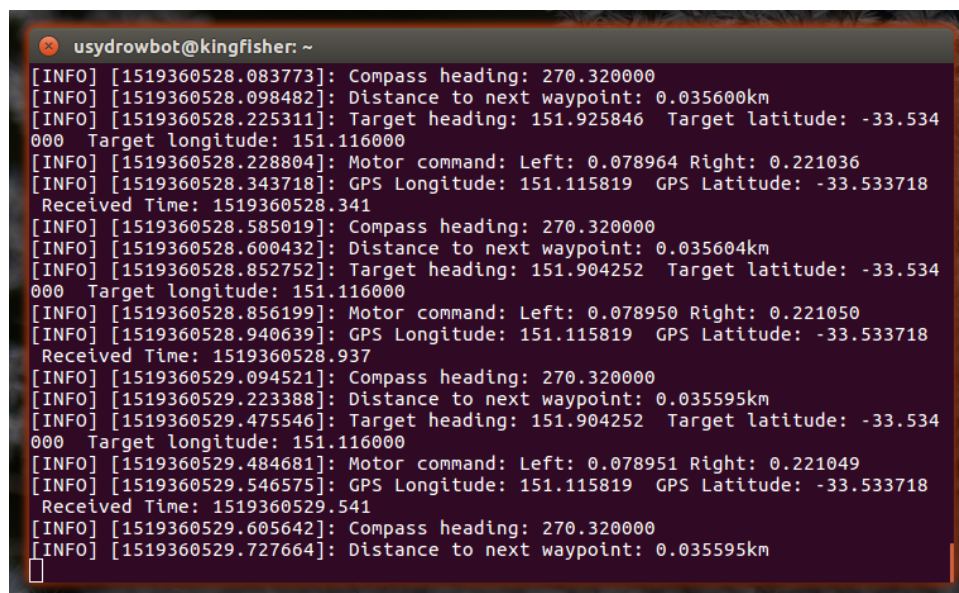
Fig 1. Core services interface

## Navigation

Autonomous navigation starts with receiving current GPS coordinate, then comparing the current position with first waypoint, if the distance is larger than threshold, a target heading to the waypoint is computed, then PID controller adjusts the throttle of each motor to reach target heading. When the target waypoint is reached, the next waypoint is set. nav.py handles ROS topic to send and receive information while nav\_core.py contains core navigation logic. nav\_core is a navigation Python class which contains all the navigation related parameters including coordinate, heading and waypoints.

Navigation program is executed by the command:

```
roslaunch kf_nav nav.py
```

A terminal window titled 'usydrowbot@kingfisher: ~' displays the output of the navigation program. The output consists of multiple lines of log messages, each starting with '[INFO]' and a timestamp. The messages provide real-time data on the robot's navigation, including compass heading, distance to the next waypoint, target heading and latitude, and motor commands for the left and right motors. The log shows a sequence of waypoints and the robot's progress towards them.

```
usydrowbot@kingfisher: ~
[INFO] [1519360528.083773]: Compass heading: 270.320000
[INFO] [1519360528.098482]: Distance to next waypoint: 0.035600km
[INFO] [1519360528.225311]: Target heading: 151.925846 Target latitude: -33.534
000 Target longitude: 151.116000
[INFO] [1519360528.228804]: Motor command: Left: 0.078964 Right: 0.221036
[INFO] [1519360528.343718]: GPS Longitude: 151.115819 GPS Latitude: -33.533718
Received Time: 1519360528.341
[INFO] [1519360528.585019]: Compass heading: 270.320000
[INFO] [1519360528.600432]: Distance to next waypoint: 0.035604km
[INFO] [1519360528.852752]: Target heading: 151.904252 Target latitude: -33.534
000 Target longitude: 151.116000
[INFO] [1519360528.856199]: Motor command: Left: 0.078950 Right: 0.221050
[INFO] [1519360528.940639]: GPS Longitude: 151.115819 GPS Latitude: -33.533718
Received Time: 1519360528.937
[INFO] [1519360529.094521]: Compass heading: 270.320000
[INFO] [1519360529.223388]: Distance to next waypoint: 0.035595km
[INFO] [1519360529.475546]: Target heading: 151.904252 Target latitude: -33.534
000 Target longitude: 151.116000
[INFO] [1519360529.484681]: Motor command: Left: 0.078951 Right: 0.221049
[INFO] [1519360529.546575]: GPS Longitude: 151.115819 GPS Latitude: -33.533718
Received Time: 1519360529.541
[INFO] [1519360529.605642]: Compass heading: 270.320000
[INFO] [1519360529.727664]: Distance to next waypoint: 0.035595km
```

Fig 2. Typical terminal display when running navigation program

Cruise throttle, PID value for heading adjustment, log speed and waypoint coordinates are set in config file. To override autonomous navigation, flip switch 2 on radio transmitter to activate manual control.

Manual control is written in high-level ROS to control the throttle and rotation with the rotation scalar configurable.

To use nav\_core.py in non-ROS environment, simply remove ROS log command in the file.

## Driver

Drivers for GPS and compass are programmed to be integrated with ROS. GPS driver is available in both LCM and ROS versions. The program reads the USB serial input from GPS and compass and then filter the value out to be converted to floats and sent to ROS topic.

The drivers can be initialized by the shell script.

## Monitor

A system monitor is programmed to check system health in the background, including radio signal, battery voltage, current and PCB temperature. The source of these information comes from MCU via serial link and accessed via ROS /sense topic. The value for warning threshold can be set in config file. The warnings will appear on the terminal as well as logged in the log file.

The monitor can be initialized by the shell script.

## Hardware

### MCU

MCU is integrated with an Arduino that controls serial link with Atom PC, link with brushless motor controller, LED light signal and radio receiver. Note low-level manual control with similar functionality to the software program developed is also offered in MCU with no configuration option. In manual mode, the MCU Arduino handles the radio PPM input and computes the output throttle to sent directly to two motor controllers.

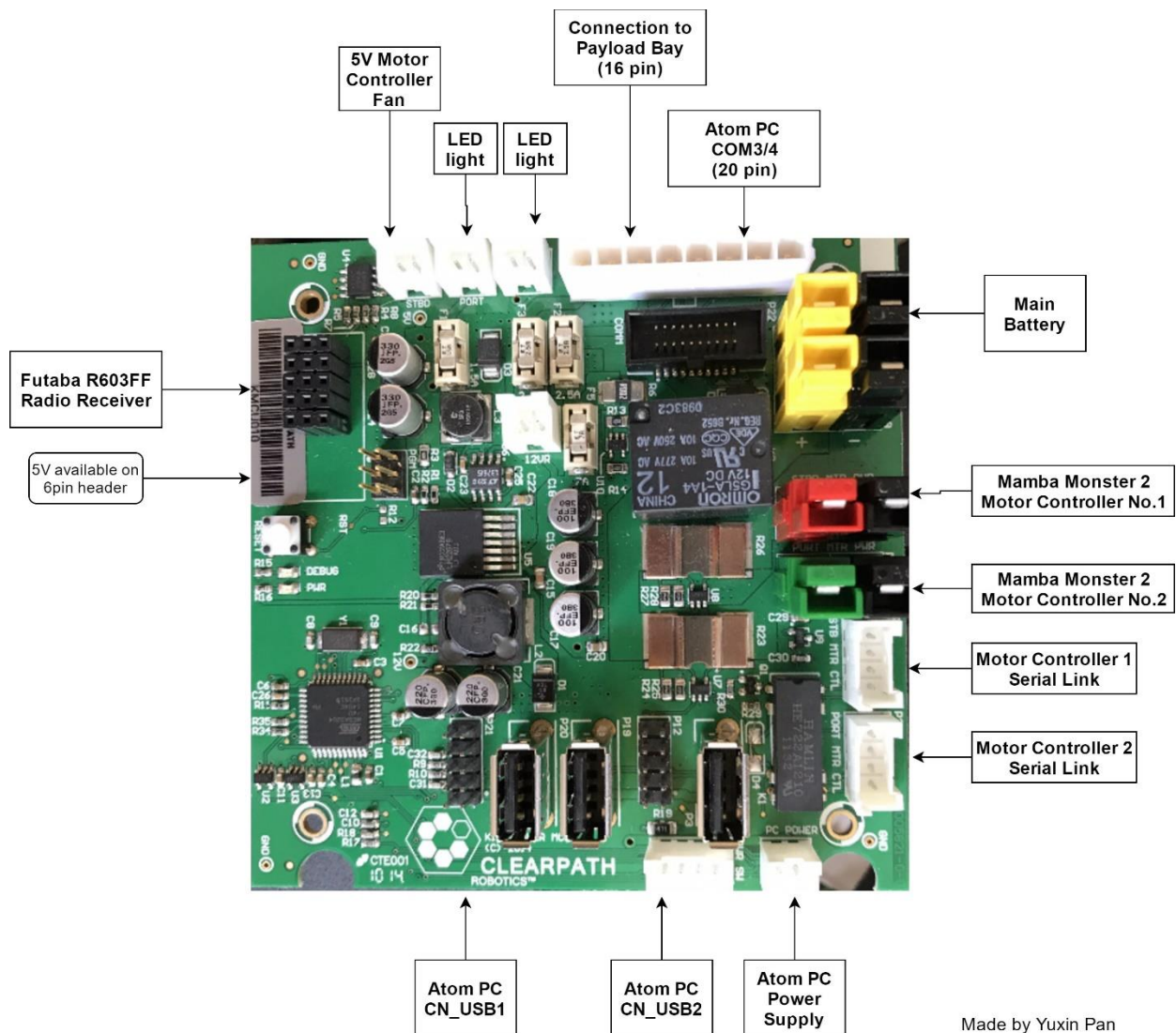


Fig 3. Wiring diagram of Kingfisher MCU

## Atom PC

Atom PC is the high level processing power on Kingfisher, the model of the PC is LE-376. In total, 4 connections are made between MCU and Atom PC:

1. CN\_USB1
2. CN\_USB2
3. COM3/4
4. DC\_IN (which connects to battery via MCU without step-down)

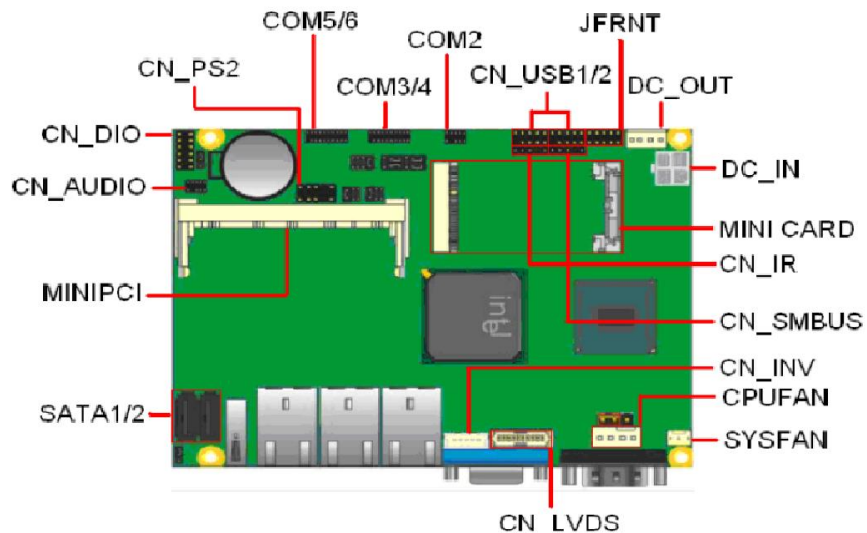


Fig 4. Atom PC LE-376 pinout

## Others

Official user manual for Kingfisher can be found in:

<https://oceanai.mit.edu/svn/moos-ivp-kfish/trunk/docs/kfish-m200-manual.pdf>

Source code (incomplete) by Clearpath is at:

<https://github.com/clearpathrobotics/kingfisher>

[https://github.com/clearpathrobotics/clearpath\\_kingfisher](https://github.com/clearpathrobotics/clearpath_kingfisher)

Support for Kingfisher:

<http://wiki.ros.org/Robots/Kingfisher>

<http://www.clearpathrobotics.com/assets/guides/kingfisher/WorkingWithKingFisher.html>