THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF MECHANICAL ENGINEERING

A COST-EFFECTIVE AUTONOMOUS ZERO-TURN MOWER FOR ORCHARDS

MICHAEL A. PAGAN
SPRING 2021

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Mechanical Engineering
with honors in Mechanical Engineering

ABSTRACT

Orchard maintenance activities can be time-consuming and potentially hazardous for humans. Therefore, it is beneficial to automate orchard tasks to remove humans from undesirable or harmful jobs. One key orchard maintenance activity is mowing between rows of trees. To automate this task, the mower must know where it is at all times, to a relatively high degree of accuracy. Achieving cost-effective, high-accuracy localization can be difficult, however.

In this project, a differentially steered unmanned ground vehicle (UGV) named UGV01 was used to develop a mission-based autonomy system. UGV01 was used as a proof-of-concept platform before implementing the system onto a Cub Cadet RZT-S zero-turn mower.

The latest high-precision satellite localization and waypoint automation technologies were tested on UGV01. A Pixhawk autopilot controller was used to achieve mission-based autonomy. High precision satellite localization was achieved through a Real Time Kinetic (RTK) positioning system established at the testing grounds. The system was built with affordable RTK boards, thereby demonstrating the ability of modern technology to provide a viable, cost-effective solution for autonomy. Robot Operating System (ROS) was integrated into the ground control system of UGV01 to enhance its flexibility and functionality. ROS was used to create an automated mission-updating routine for UGV01. A simple object-avoidance routine was created to demonstrate the sensor-integration possibilities of ROS.

The Cub Cadet zero-turn mower was modified to be compatible with the Pixhawk system developed for UGV01. A custom RS485 interface interpreted the outputs from the Pixhawk system to successfully control the drive wheels and mowing deck on the Cub Cadet.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

My completion of this thesis was made possible by the unending enthusiasm, mentorship, and brilliance of Dr. H.J. Sommer. Through this fun and challenging project, Dr. Sommer helped me find my passion for autonomous ground vehicles. I cannot thank him enough for his contributions to this project and support of my development as a researcher and engineer.

I am very grateful of Randall Bock and all of his help with retrofitting the Cub Cadet RZT-S Zero for autonomous operation. Randall was always eager to lend a hand and his advice was critical in several design decisions.

My work with ROS would not have been possible without the help of Chris Hirsh. Chris went out of his way to do a timely rebuild of the Linux PC, allowing for important project progress over winter break.

As always, I would like to thank my parents and family. My successes are undoubtedly rooted in their love for me and their encouragement of excellence.

Chapter 1

Motivation

Machinery operations in orchards such as mowing the rows between trees, spraying, etc. are low-skill, time-consuming, and hazardous. Autonomous machinery is able to remove humans from these tedious and dangerous tasks, while completing them with greater efficiency. Consequently, there is a great deal of benefit to be realized from integrating autonomy into orchard operations.

While the agriculture industry has much to gain from the adoption of autonomous machinery, high-precision autonomous guidance systems in agriculture have come at a high price tag for many years, making them unavailable to smaller-scale orchard farmers. Advances in technology have allowed for low-cost, high-precision GPS systems to become available to consumers. The emergence of this technology provides an opportunity to make mission-based autonomy more widely available to orchard farmers.

This project aims to build an autonomous guidance system for an all-electric Cub Cadet zero-turn mower using an affordable high-precision GPS system. The performance of this system will gauge the feasibility of retrofitting a factory-built mower for autonomous operation with current technology. If some degree of success is achieved, further development and research may lead to widespread adoption of autonomous orchard machinery in the near future.

Chapter 2

Literature Review

2.00 Overview

This chapter reviews published research relevant to the development of an autonomous orchard mower. It also provides key details on concepts related to the project work. Research has been completed that demonstrates the efficiency increases that autonomy brings to agricultural applications. Agricultural autonomy has been tested with and without the use of GPS. The benefits and drawbacks of each approach are discussed.

2.01 Satellite-Based Localization

Before reviewing research related to agricultural autonomy, it is worth explaining the technology that allows autonomous vehicles to know where they are in space (i.e. localization). Perhaps most relevant to outdoor autonomous ground vehicles is satellite-based localization. The Global Positioning System (GPS) is a satellite localization service owned and operated by the United States Government. Although GPS is the oldest and most widely used satellite system, it is just one of four that make up the Global Navigation Satellite Systems (GNSS). The others include BeidDou (China), Galileo (European Union), and GLONASS (Russia) [1]. Using any one of the systems alone, however, results in precision and accuracy shortcomings.

A satellite receiver finds its position on Earth by calculating its distance from at least three satellites. This distance is calculated by multiplying the time it takes for signals from each satellite to arrive at the antenna by the speed of the signal (the speed of light). Every GNSS satellite transmits signals in at least two frequency bands, L1 and L2, the latter being higher in frequency. Almost all civilian devices use only the L1 band, whereas military and robust commercial devices use both L1 and L2 bands. These dual-frequency GPS antennas are able to correct for atmospheric distortions and improve accuracy. In a similar way, receiving signals from more satellites improves positional accuracy. The best way to accomplish this is to use a GNSS antenna capable of harnessing all four satellite systems [2]. On the frequencies of the L1 and L2 bands, satellites communicate pseudo-random codes. When a receiver uses these codes to find its distance from the satellites, it performs code-phase calculations. Since the pulse width of the codes is relatively long, in other words the frequency is low, the position estimate is, at best, within 10 ft. Carrier-phase calculations use the unmodulated L1 and L2 waves to find the distance to a satellite [3]. Since the L1 and L2 frequencies are much higher than those of the codes, positional accuracy can be as good as a few millimeters [4]. In general, dual-band, carrier-phase, GNSS devices are limited to large-scale, high-capital applications due to the historically high cost and physical size of the positioning systems [2].

To further improve the accuracy of satellite positioning, local corrections can be made to account for errors in satellite signals. Local corrections are categorized into Differential GPS (DGPS) and Real Time Kinematic (RTK) GPS. Both correction techniques recognize the variance of satellite errors and aim to correct them in real time. This is done by receiving satellite data at a base station with a precisely known location. The calculated location is compared with the known location and position correction signals are sent over radio transmitters. Rovers of unknown location can then receive these signals to correct their satellite data and achieve a more accurate position solution. The difference between DGPS and RTK GPS is routed in the method of position calculation: DGPS uses code-phase calculations whereas RTK GPS uses carrier-phase calculations, as shown in Figure 2.1. Expectedly, DGPS is less accurate (+/- 1m) and slower, but its radio transmissions have less data and can be utilized far away from the base station (100-200 km). RTK GPS provides a dynamic accuracy of a few centimeters. Its radio transmissions are fast and precise but require a large amount of data. Additionally, the rover is constrained to a smaller operating radius from base station (10-20 km) [5]. Following the trend of accuracy and cost, RTK GPS is historically one of the most expensive satellite localization systems [6].

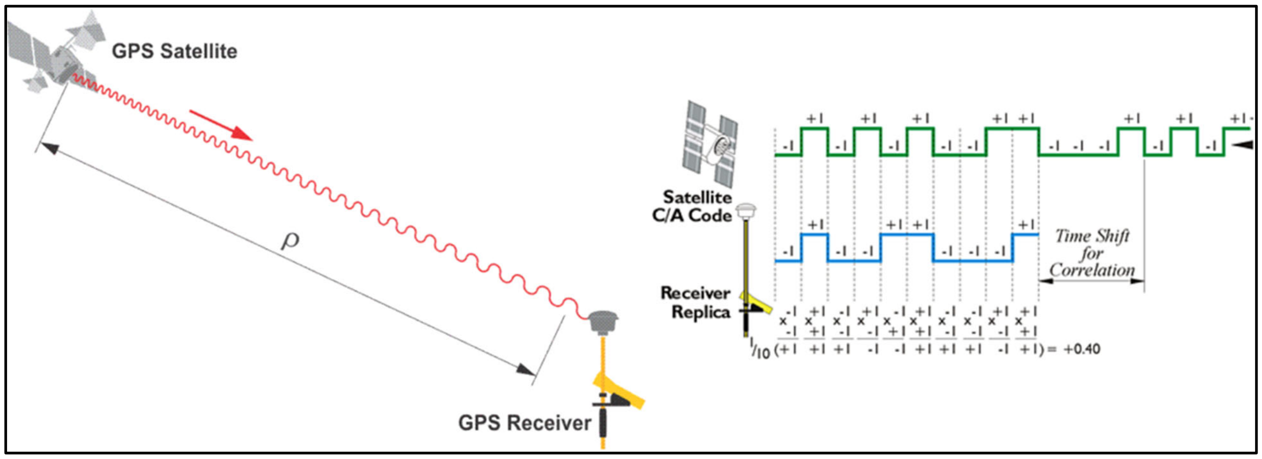


Figure 2.1 - Comparison of carrier-phase (left) and code-phase (right) calculations by a GPS receiver [3]

2.02 Sensing-Based Localization

Other sensing instruments are common on autonomous vehicles to increase safety and vehicle awareness. When GPS is unreliable or unavailable, sensory data can be used for both localization and mapping. Simultaneous Localization and Mapping (SLAM) uses sensor input to map the surrounding environment, learn the map, and identify its position within the map [7]. Laser ranging with a Light Detection and Ranging (LiDAR) module is commonly used as the sensor input for SLAM. LiDAR emits lasers pulses that reflect off surrounding surfaces. Upon receiving the reflected light, the distance to those surfaces is measured. A singular, stationary beam pointed along an axis will produce a 1D scan, a singular laser sweeping along a plane will produce a 2D scan, and several lasers distributed along the vertical axis sweeping across a horizontal plane will produce a 3D scan [8]. Ultrasonic sensors can also be used for ranging, but they perform worse than laser ranging. Ultrasonic sensors emit high-frequency sound waves that reflect off surrounding surfaces and return to the sensor. Vision-based methods for sensing the environment are also available. Machine vision processes the images taken by a camera, relying heavily on computing to process the images and identify objects in its surroundings [9].

2.03 Unmanned Ground Vehicle Path Planning Using Multiple Sensor Inputs

An unmanned ground vehicle (UGV) is a ground vehicle that operates without an onboard, human operator. UGVs can operate autonomously or by remote control. Autonomous UGVs take sensor input, decide on the safe path forward, and activate the motors accordingly. In one study by Rawashdeh and Jasim [10], the UGV shown in Figure 2.2 was able to safely navigate a clearly delineated grass path with unexpected obstacles by using multiple sensor inputs. The sensors used to detect the lines and obstacles were machine vision, a digital compass, a GPS receiver, and LIDAR. The input data from each of these sensors was fused into a cost matrix to determine the lowest cost path. Detected obstacles were assigned positive cost values and desired headings were assigned negative cost values. The lowest cost path was the safest route to follow [10]. Multiple sensor inputs can be utilized to improve the performance of autonomous vehicles.

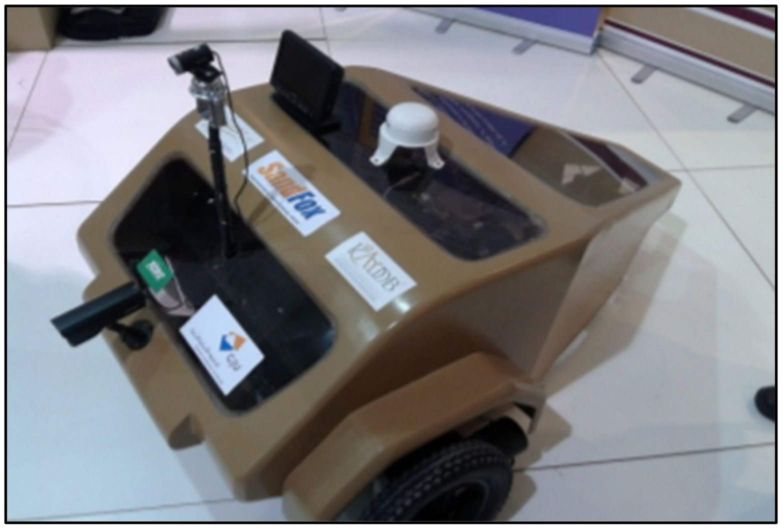


Figure 2.2 – Example of a UGV with multiple sensor inputs [10]

2.04 Precision Agriculture with Autonomous UAVs

The promise of augmenting agricultural efficiency with autonomous vehicles can be seen in the use of unmanned aerial vehicles (UAVs) for field spraying operations. Similar to a UGV, an unmanned aerial vehicle can be controlled remotely or autonomously. The methods by which unmanned aerial vehicles are automated can be applied to ground vehicles. Unmanned aerial vehicles, however, have an additional spatial parameter of concern—altitude—that can be ignored for UGV autonomy. In one study, a 3-quadcopter crop-spraying simulation was carried out using two different mission assignment programs. A "mission plan" refers to the path plan mapped before an autonomous flight. Missions consist of many waypoints, which are latitude-longitude-altitude locations to be reached in sequence. By pre-calculating an optimal mission plan, the spraying time was significantly reduced, regardless of the number of quadcopters or size of field [11]. From this simulation study, it is evident that mission planning for autonomous vehicles allows for performance optimization and consequent overall efficiency increases in agriculture.

2.05 Autonomous Agriculture Vehicle Guidance with GPS and Path Planning

Mission planning for autonomous agriculture vehicles requires the use of GPS during operation. In order to follow the pre-mapped route, the vehicle must be able to locate itself in space. To study the effect of a mission planner on the efficiency of an autonomous tractor, Bochtis, Vougioukas, and Griepentrog developed a mission planner to generate an optimal path for mowing or spraying operations in a field [12]. When an autonomous tractor with RTK GPS utilized the path developed by the mission planner, the researchers found that non-working time during one or multiple-field operations was significantly reduced. The high-level mission planner is effective in achieving maximum efficiency by determining the optimal path.

In a subsequent study, Bochtis, Vougioukas, and Griepentrog applied their mission planning methods to orchards [13]. Orchards provide a unique and optimal opportunity for mission planning due to the unchanging position of tree rows. A planned route for an autonomous orchard machine is long-lasting as the desired path will be consistent year to year. In the study, optimal path plans for single and multi-row mowing and spraying operations were generated for an autonomous tractor with RTK GPS. The optimal operation plans allowed the autonomous machine to reduce non-working time by up to 32.4% and non-working distance by up to 40.2%, when compared to the working time and distance of conventional, non-optimized orchard machine operation, as depicted in Figure 2.3 [13].

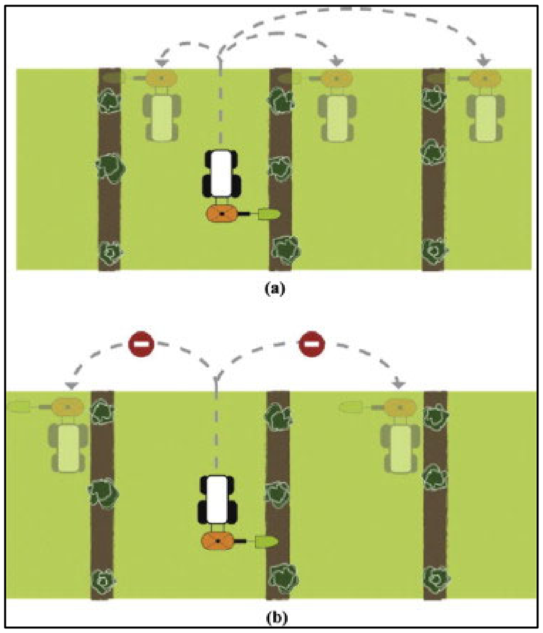


Figure 2.3 – Orchard mowing efficiency is improved with mission planning [13]

To effectively utilize the efficiency provided by a mission planner, autonomous tractors must be able to accurately follow the path. Precision and accuracy are the two error types concerning GPS quality [14]. Precision is the repeatability of positioning; i.e. how close a position measurement is to the previous position measurements. Accuracy is how close the measured position is to the actual position on a map. For tractor guidance systems that require a human operator on-board, precision is of primary importance because the tractor simply follows relatively short, straight, adjacent paths. Low-cost GPS systems can only provide 10-meter accuracy but their precision is less than a meter. The precision is far more acceptable than the accuracy, but quantization error can cause precision deviations greater than 0.1 meter. A study was done to improve precision of low-cost GPS receivers with a Kalman filter [14]. The Kalman filter is a mathematical algorithm that, after taking input data, generates an estimate based on prediction and observation models. Employing the Kalman filter on a low-cost GPS receiver decreased quantization error by 43% and standard deviation of heading angle by 75%. Overall, the use of a Kalman filter with low-cost GPS increases localization precision and smooths vehicle trajectory.

GPS precision error in autonomous agricultural machines must also be considered for the implements of the machines. An implement is a piece of equipment attached to the rear hitch of the tractor. An implement is used to perform operations such as plowing, mowing, baling hay, etc. An autonomous tractor will not have a perfectly smooth trajectory and, consequently, lateral implement deviations will occur. In one study, the deviations of an implement on an RTK GPS autonomous tractor were recorded as shown in Figure 2.4 [6].

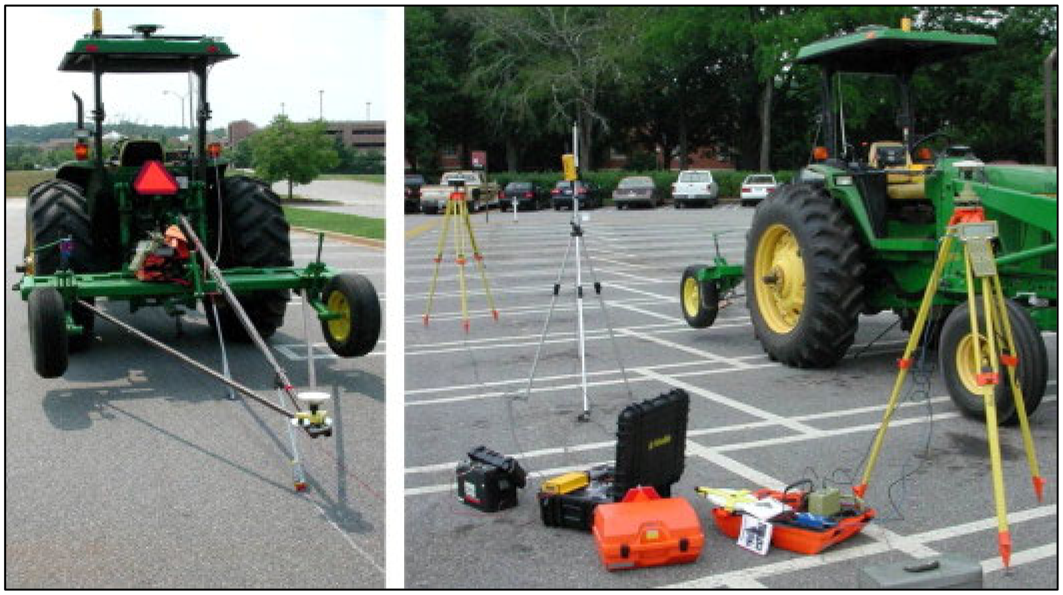


Figure 2.4 - Test rig to record the lateral deviation of an implement [6]

Three implement positions were tested: at the rear axle, 180 centimeters from the rear axle, and 360 centimeters from the rear axle. The largest root-mean-square extreme lateral deviations occurred at 7.2 kilometers per hour (highest tested tractor speed) with an implement mounted 360 centimeters from the rear axle. This root-mean-square deviation was 5.2 centimeters. Slower tractor speeds and closer mounted implements had lower root-mean-square extreme deviations. The use of longer implements on GPS-guided autonomous tractors sacrifices localization accuracy and therefore efficiency.

Mission planning and RTK GPS have been used in an orchard-mowing autonomous tractor system. John Deere researchers conducted a field test of autonomous tractors over several months using LiDAR and cameras for obstacle avoidance and RTK GPS for localization [15]. A supervisor gave mowing tasks to the tractors in a citrus orchard and addressed difficulties when the tractors were unsure of how to safely proceed. The tractors increased orchard maintenance productivity by 30%. Time operating at full speed was chosen as the indicator of productivity. In manually operated tractors completing the same mowing tasks, maximum speed range is held for less than 5% of the total working time. The autonomous tractors in the study operated in the maximum speed range for 65% of the time. Optimal path planning also positively impacted productivity and efficiency. Based on the number of acres covered in a day (a direct result of higher average speed) the autonomous tractors were 30% more productive than manually driven ones.

2.05 Autonomous Orchard Navigation without GPS

Autonomous vehicles that rely on traditional GPS will run into issues with positional accuracy. Instead of augmenting GPS data with other sensor inputs, researchers have attempted GPS-free methods of autonomously navigating orchards. A 2D laser scanner has been used to successfully guide an unmanned tractor through a row of trees [16]. The test was performed specifically to determine feasibility within orchards, identifying the difficulties of GPS usage under large tree canopies. The 2D laser identified tree trunks such that the surrounding tree rows could be mapped. Using the 2D laser scanner (with calibration and noise removal) as the sole sensing instrument, the unmanned tractor could navigate the tree row in real time with a lateral and angular heading mean error of 0.11 meters and 0.36 degrees, respectively. Although the autonomous tractor was successful, a significant limitation must be noted: for the tractor to be successful, its speed had to be 0.36 meters per second (< 1 mile per hour). For real-world orchard applications, this speed would be highly inefficient, making adaptation of the technology unrealistic [16].

In a more real-world application, a GPS-free, all-electric utility vehicle was operated autonomously in an orchard as shown in Figure 2.5. Laser range sensors were used to detect and model the rows of trees. The vehicle was able to autonomously navigate eight, 3-meter-wide orchard rows. The machine would turn at the end of a row, find the next row with its laser range sensors, and proceed into that row. It should be noted that this vehicle could only pass down the middle of the row. Canopy and trunk size influenced performance: large canopies obstructed foresight into the next row and small trunks made row-identification difficult [17]. In a study done to improve the performance of an autonomous orchard vehicle operating without GPS, wheel and steering encoders were added. The data from the wheel and steering encoders fed a path-tracking controller which helped improve the smoothness of turning [18].

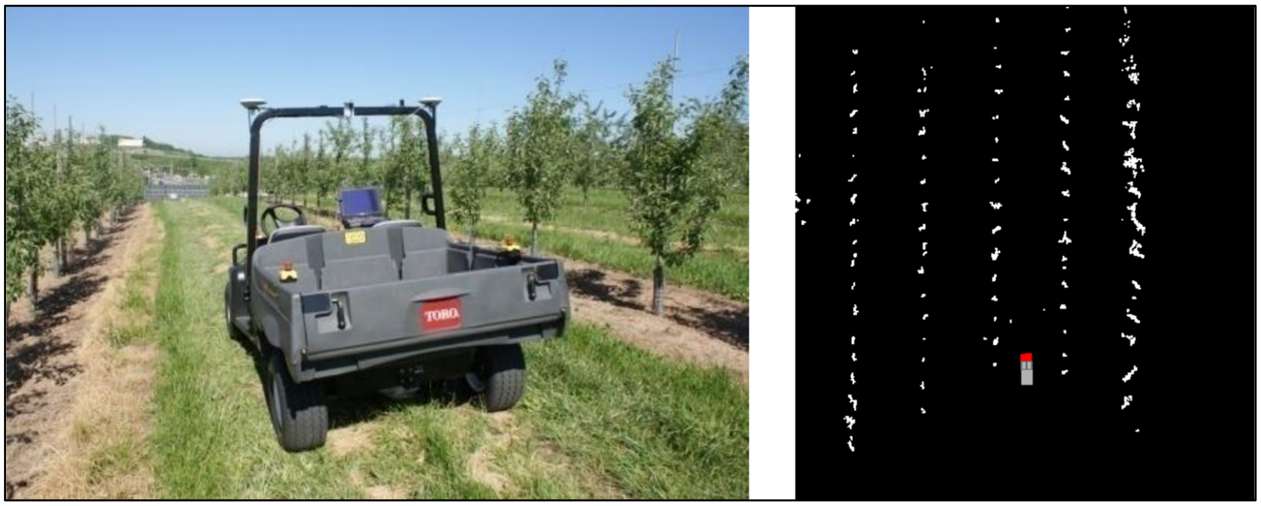


Figure 2.5 – Laser-based autonomous orchard vehicle (left) navigating rows while mapping tree locations (right) [17]

Machine vision has been tested as an alternate method of vehicle localization for autonomous UGVs operating without GPS. Radcliffe, Cox, and Bulanon used a multispectral camera and processing computer to detect tree canopies with the sky as the background [19]. Most machine vision applications in orchards aim to detect trunks and canopies, looking forward with the ground in view. In the sky-based detection study, an autonomous UGV differentiated the tree canopies from the sky and centered itself between the canopies. Using only this machine vision technique as its only means of localization, the UGV was able to navigate down the middle of an orchard row with a root-mean-square center deviation of 2.13 centimeters. There are two main limitations to this method of localization in orchards. The first is that the size of tree canopies affects center deviation. The other is that the sky-based imaging becomes useless at the end of an orchard row. When the canopies are no longer visible, the UGV has no usable sensor input.

2.06 A Cost Effect Autonomous Zero-Turn Mower for Orchards

The focus of this research was to design, build, and evaluate a cost-effective autonomous zero-turn mower for orchards. A zero-turn mower does not operate like a conventional tractor, i.e. Ackerman steering (turning the front wheels). Rather, the rear drive wheels are independently controlled by the operator (differential steering) and the front wheels are free-spinning casters. An all-electric Cub Cadet RZT-S Zero mower (shown in Figure 2.6) was modified to operate as an autonomous differential-steer UGV.

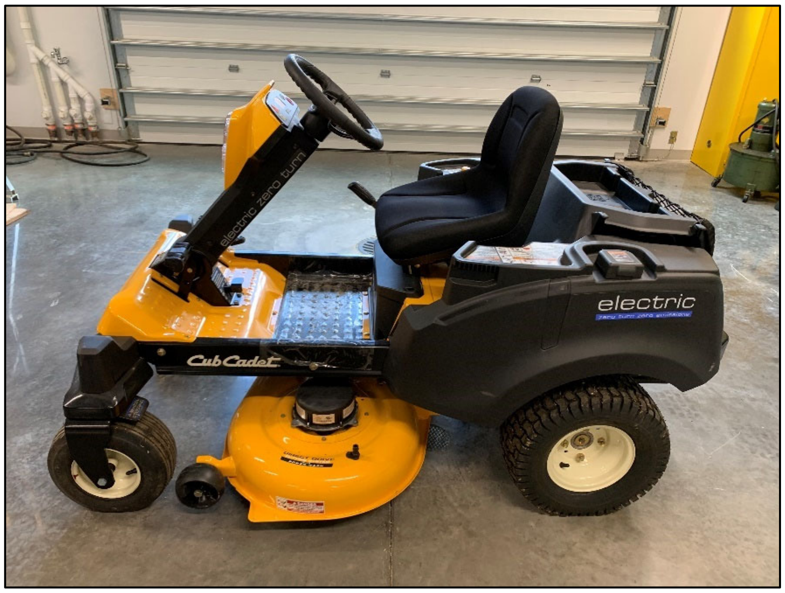


Figure 2.6 - Unmodified Cub Cadet RZT-S Zero

One objective was to determine if current technology allows a cost-effective autonomous control system to be successful on the retrofitted machine. Several previous agricultural autonomy studies have been done without cost as a focus, namely those utilizing expensive RTK GPS [6,12,13,15]. Studies that sought to eliminate the cost of precise GPS-based localization were unable to utilize the efficiency benefits provided by mission planning. Additionally, environmental factors (canopy, trunk size, etc.) were detrimental to the performance machines without GPS localization [16–19]. To capitalize on the proven efficiency of mission planning, this design includes necessary satellite-based localization. Fortunately, recent technological advancements have allowed low-cost RTK systems to become available in the satellite positioning market. Due to their novelty, these systems have not yet been adopted for agricultural use. This presents an opportunity to capitalize on low-cost, high precision localization in order to achieve cost-effective autonomy in an orchard. In pursuing the highest precision possible, it should be noted that the relatively small footprint of a zero-turn mower with the deck mounted under the chassis reduces the magnitude of accuracy deviations during operation, increasing tolerance for GPS error [6]. Improved safety and performance can be achieved through additional sensing equipment (LiDAR, ultrasonic sensors), but this will also increase cost [10]. By utilizing a low-cost, high-precision RTK GPS system and supplementary sensors, an efficient low-cost autonomous orchard mower was designed.

Chapter 3

Remote Control of a Differential Steer UGV

3.00 Overview

This chapter describes the remote-control differential steer ground vehicle built as a scaled test prototype. For testing and proof of concept of automation technology, it is more feasible to utilize a scaled machine. The Cub Cadet RZT-S is a differential steer vehicle, thus the control system designed for the scaled ground vehicle can be ported to the Cub Cadet after development and testing.

3.01 Differential Steer UGV

Project work began with a custom, battery-powered differential-steering UGV named UGV01. Plastic tank tracks run along the length of UGV01, each guided by a front and rear sprocket. Suspension for each track is provided by a spring coupling a pair of road wheels as shown in Figure 3.1.

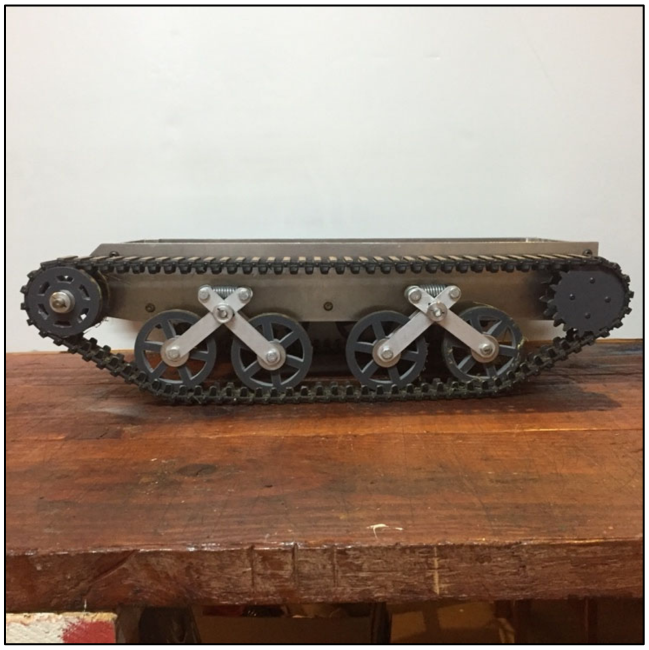


Figure 3.1 - UGV01 left side showing track design

Each track is driven by an 18V Dewalt DW960 right angle drill motor (shown in Figure 3.2) controlled by a Sabertooth 2x60 motor controller (not shown in Figure 3.2). This controller accepts PWM, PPM, and serial signals, operating the motors at 6-30 V with a 60 A maximum current output. UGV01 is powered by a 12V 9Ah gel cell battery. A heavy-duty switch mounted on the aft of the vehicle connects the battery to the motor controller.

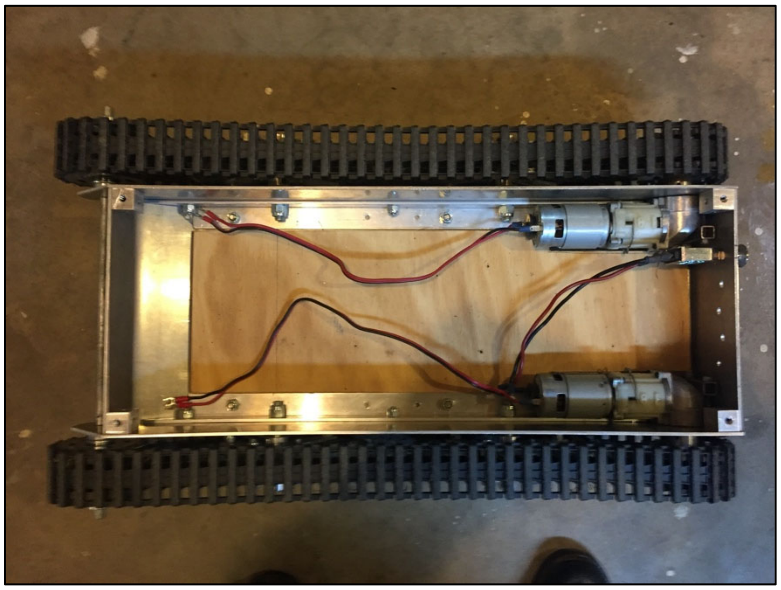


Figure 3.2 - UGV01 top showing motors and aft switch

3.02 Remote Control Configuration

In order to remotely control UGV01, a Spektrum R/C system was configured with the Sabertooth motor controller. This system uses a Spektrum DX8 G2 Transmitter, Spektrum AR8010T Receiver, and SPM9645 DSMX Remote Receiver. The simplest transmitter input method for controlling a differential steering vehicle is two sticks that spring back to the center, such that the resting outputs are neutral signals. The right stick on the Spektrum DX8 springs to center and, by default, up/down controls channel 3 and left/right controls channel 2. Since Spektrum R/C components are designed for aircraft, common channels are labeled by the aircraft component they usually control. For example, channel 2 (AIL) is for the aileron, and channel 3 (ELE) is the elevator as shown in Table 3.1. Channel assignment can be customized on the DX8. More detail is provided in Section 4.02.

Table 3.1 - Spektrum DX8 channel configuration for R/C UGV01

DX8 Channel	*Assignment*	*UGV01 Function*
1 (THR)	Throttle (left stick up/down)	N/A
2 (AIL)	Aileron (right stick right/left)	Steering
3 (ELE)	Elevator (right stick up/down)	Throttle
4 (RUD)	Rudder (left stick right/left)	N/A
5 (GER)	Switch A	N/A
6 (AUX1)	Switch D	N/A
7 (AUX2)	Right Knob	N/A
8 (AUX3)	Right Knob	N/A

Signals are sent from the DX8 transmitter, captured by the DSMX remote receiver, relayed to the AR8010T receiver, and sent to the respective channel pinouts as Pulse Width Modulated (PWM) signal. Channel 2 (AIL) and channel 3 (ELE) of the receiver are connected to the S2 and S1 terminals of the Sabertooth, respectively, as shown in Figure 3.3.

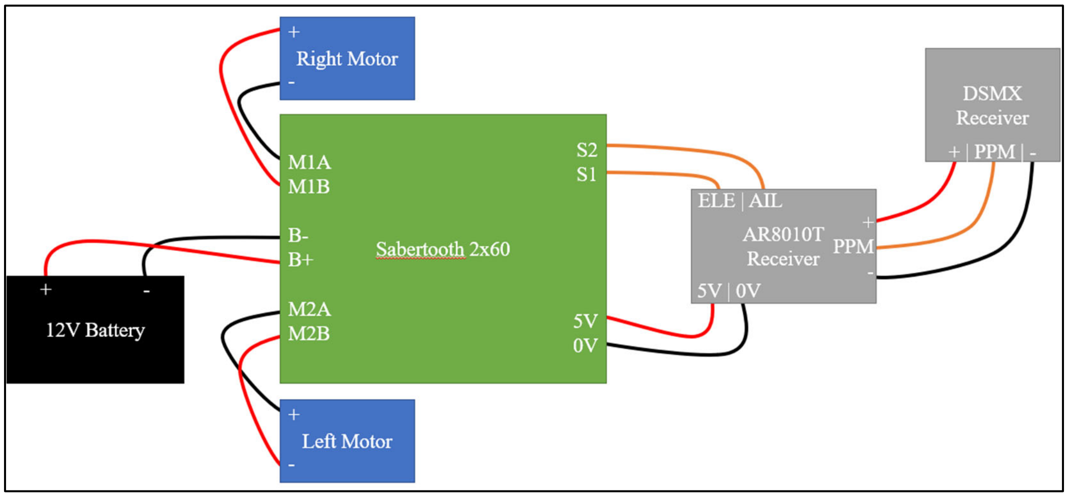


Figure 3.3 - Wiring configuration for R/C system in UGV01

For UGV01 to move forward in a straight line, the left motor must rotate clockwise and right motor counterclockwise. Both motors are identical, so applying a positive voltage to a positive lead will cause clockwise rotation. With mixing enabled on the Sabertooth, a forward throttle command sends positive voltage to M1A and M2A. To achieve counterclockwise rotation of the left motor, its positive lead must be wired to M2A and negative lead to M2B.

When working with R/C signals, an important distinction must be made between Pulse Position Modulation (PPM) and PWM. In the UGV01 R/C configuration, both signal types are present, as shown in Figure 3.4.

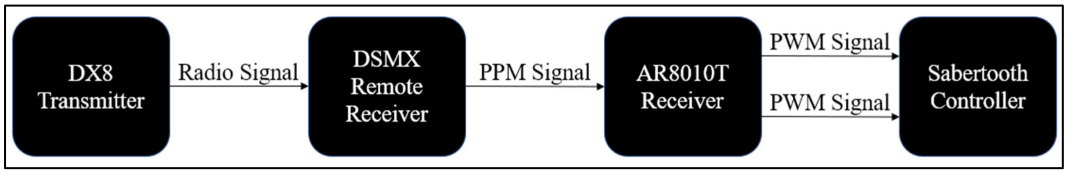


Figure 3.4 - Signal flow for R/C system in UGV01

The radio signal containing commands for eight channels is picked up by the DSMX Remote Receiver. The information for all eight channels is sent as PPM signal on a single wire to the AR8010T. The purpose of PPM is to transmit multiple PWM signals on one wire, as shown in Figure 3.5. This is done by spacing short pulses such that the distance between the two leading edges is the width of one PWM pulse. Therefore, to communicate eight unique PWM signals, nine PPM pulses are needed. In general, the PPM frame is 20 milliseconds long and the maximum width for a PWM pulse is 2 milliseconds, or 2000 microseconds. Since the PPM frame is 20 milliseconds, the refresh rate for each PWM channel is 50 Hz and maximum duty cycle for each channel is 10%. At any given time, only one PWM channel is at a high voltage to prevent overloading the power source.

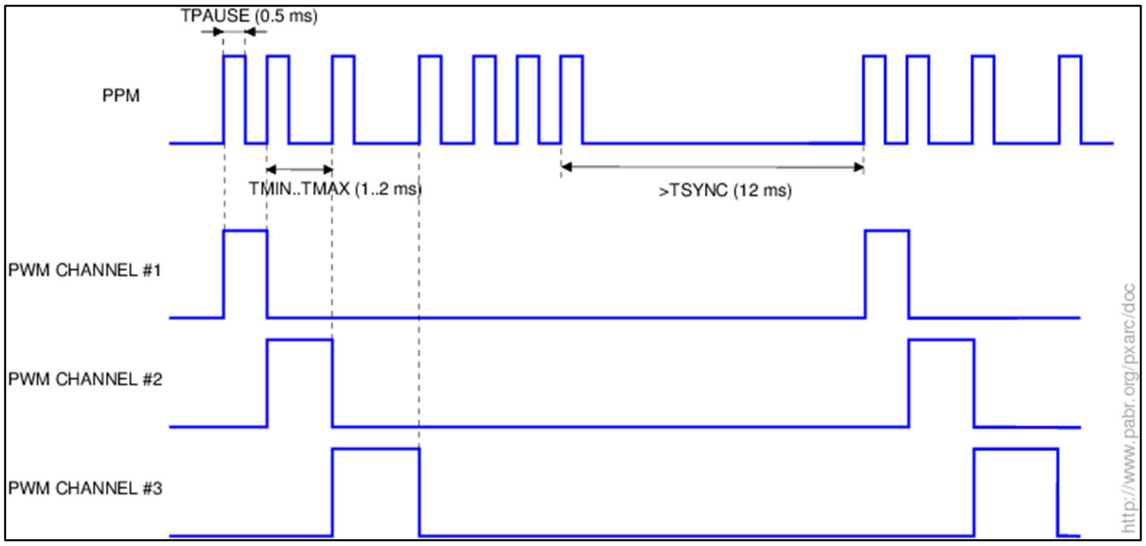


Figure 3.5 – Conversion between PPM signal and 3 PWM channels [20]

To move UGV01 correctly, the Sabertooth must be configured to properly control the left and right motor speeds. The Sabertooth is set to Mode 2, as shown in Table 3.2, to accept the PWM R/C signals from the AR8010T receiver. Mixing is enabled so that the S1 signal (up/down on right stick) controls forward/reverse motion and S2 (left/right on right stick) controls turning. If mixing was disabled, up/down would control the right motor, and left/right would control the left motor. Motor response is set to exponential to reduce the effects of UGV01's rapid turning rate.

Table 3.2 - Dip switch positions on Sabertooth for R/C configuration

Switch	***Position***	***Function***
1	Down	Accept PWM R/C signal
2	Up	Accept PWM R/C signal
3	Up	Motors are powered by a non-Lithium battery
4	Up	Mix S1 and S2 signals: ELE controls throttle and AIL controls steering
5	Down	Exponential throttle response
6	Up	0-5V signal input range

Chapter 4

UGV01 Pixhawk 4 Integration

4.00 Overview

This chapter describes the integration of the Pixhawk 4 Autopilot into the remote-control system for UGV01. A detailed description of the hardware and software configurations is provided.

4.01 Hardware Configuration

Autonomous operation of UGV01 required that it to know approximately where it is in space, where it is going, and how it will get there. For UGV01 to have this intelligence, it needed to be retrofitted with an onboard autopilot system. The Pixhawk 4 Autopilot was selected for UGV01 due to its stability, flexibility, and robustness. The Pixhawk 4 Autopilot board consists of a powerful Flight Monitoring Unit (FMU), two accelerometers, two gyro sensors, a barometer, and a magnetometer (compass). The Pixhawk 4 Autopilot also has an external module containing a GPS/GLONASS L1 antenna and integrated magnetometer. The brain of the Pixhawk 4 comes from its software: ArduPilot. When given a mission, ArduPilot takes in sensor data, assesses current location and trajectory, calculates desired trajectory, and sends out the appropriate R/C signal to motors to move toward each waypoint.

The Pixhawk needed to be integrated into the control system of UGV01 such that the Pixhawk ultimately controlled the PWM signal being sent to the left and right motors, as shown in Figure 4.2. While control of UGV01 via the DX8 transmitter would still be possible in manual mode, Pixhawk needed to be a “gate keeper” for all signals being sent to the motors. With this configuration, autonomous operation (auto mode) would not require any preceding R/C signal.

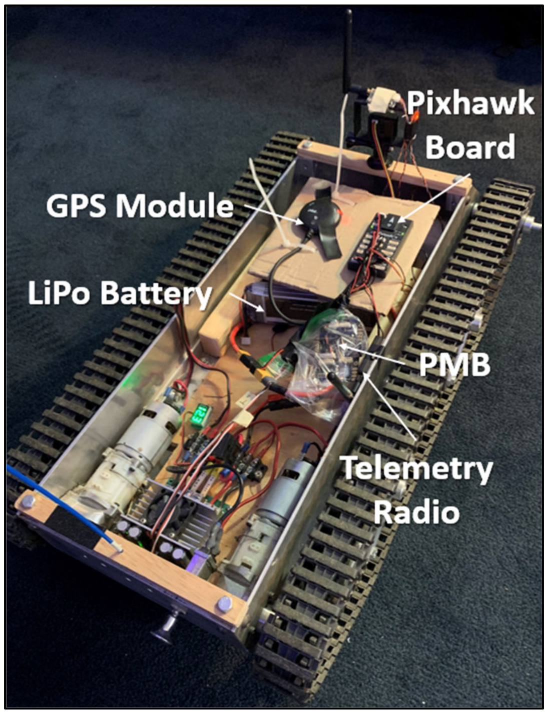


Figure 4.1 - UGV01 with basic Pixhawk system installed

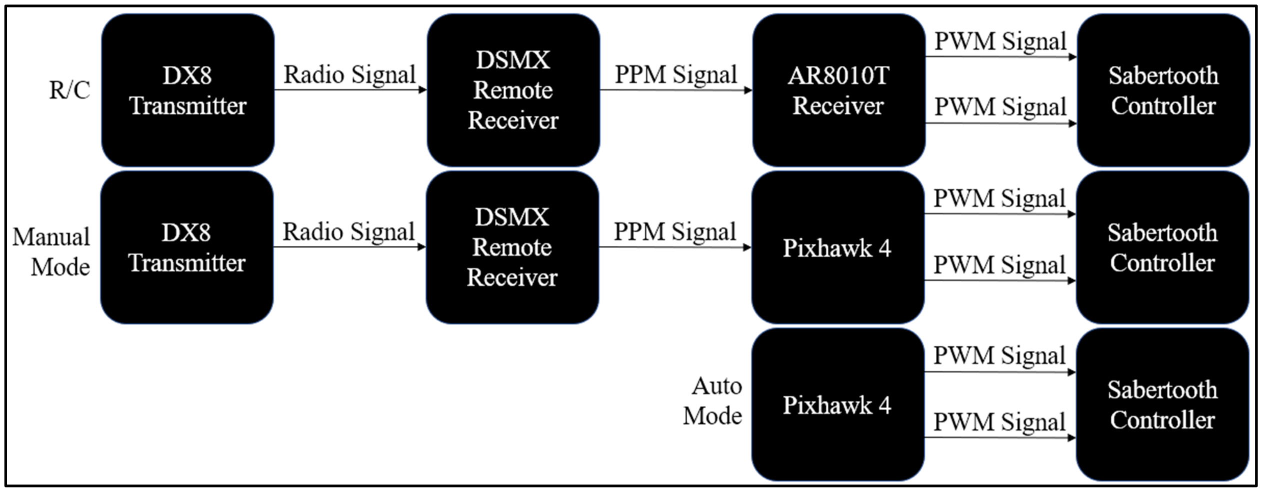


Figure 4.2 - Signal flow comparison for UGV01 with Pixhawk

To achieve the desired signal flow shown in Figure 4.2, the Pixhawk hardware components need to be properly wired into the existing UGV01 R/C system. The GPS module, DSMX receiver, LiPo battery, telemetry radio, and power management board (PMB) are connected to the ports shown in Figure 4.3. UGV01 with the Pixhawk hardware installed is shown in Figure 4.1.

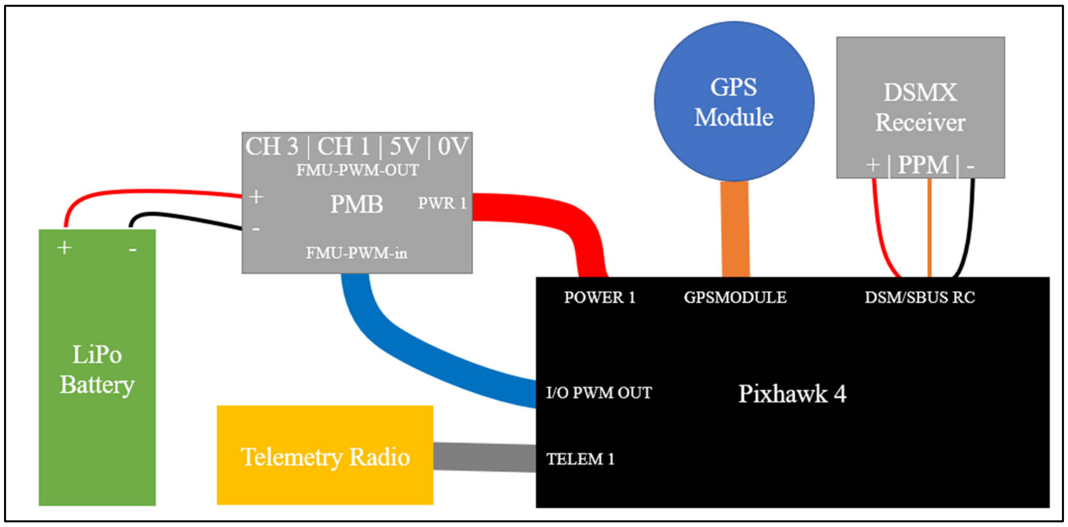


Figure 4.3 - Pixhawk hardware connections

When ArduPilot is configured for a differential-steering rover, the PWM signal for the left and right tracks are sent on separate channels (more detail on the software configuration of Pixhawk is provided in Section 4.02). If commanded to throttle forward, Channels 1 and 3 carry a PWM signal width greater than neutral 1500 microseconds. The Sabertooth interprets this and outputs a positive voltage of proportional magnitude on M1A and M2B, as shown in Figure 4.4. For the left motor to spin clockwise, M2B must be wired to the positive lead. For the right motor to spin counterclockwise, M1A must be wired to the negative lead.

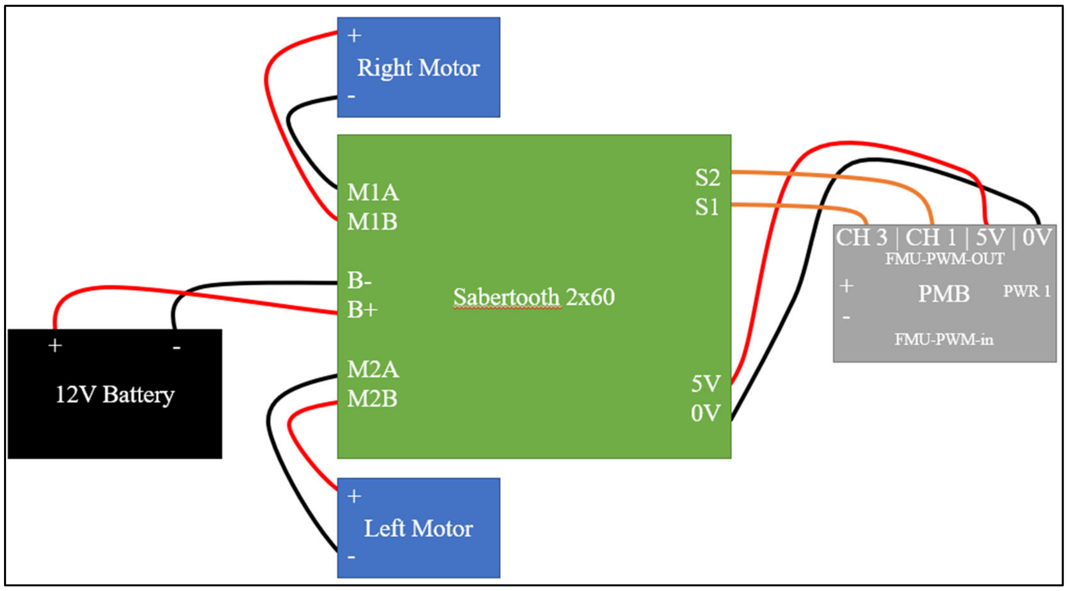


Figure 4.4 - Power management board connection to the Sabertooth motor controller

Since the Pixhawk sends PWM to the right and left tracks on separate channels (3 and 1, respectively), the dip switches on the Sabertooth must be changed to accommodate this control scheme as shown in Table 4.1. The Sabertooth no longer needs to mix signals because this is done by ArduPilot in the Pixhawk.

Table 4.1 - Dip switch positions on Sabertooth for Pixhawk configuration

Switch	***Position***	***Function***
1	Down	Accept PWM R/C signal
2	Up	Accept PWM R/C signal
3	Up	Motors are powered by a non-Lithium battery
4	Down	No mixing: S1 controls right motor and S2 controls left motor
5	Down	Exponential throttle response
6	Up	0-5V signal input range

4.02 Software Configuration

The first step in configuring the Pixhawk software is installing a ground control station (GCS) software on a Windows PC. Mission Planner was selected as the GCS for UGV01 due to its extensive support and compatibility of the ArduPilot software. Through Mission Planner, the latest firmware for Rover (i.e. the firmware designed for ground vehicles) was loaded onto the Pixhawk board. The firmware used throughout this project was Rover V4.0.0. The GCS is designed to wirelessly communicate with Pixhawk during operation. To achieve a wireless connection, a telemetry radio pair is needed. For UGV01, a Holybro 500mW telemetry radio pair was used. Out of the box, one radio was connected to the Pixhawk (as shown in Figure 4.3) and the other to the GCS computer. The GCS and Pixhawk use MAVLink (Micro Air Vehicle Link) serial protocol to communicate over the radio connection. More detail on serial communication and MAVLink can be found in Sections 6.01 and 8.03, respectively.

Within the ArduPilot firmware, there are many parameters that allow the autopilot system to be tuned for optimal performance. Mission Planner provides a user-friendly interface for configuring these parameters. The full parameter list can be found within Mission Planner under the “CONFIG” tab. ArduPilot online documentation provides helpful guides for configuring parameters to get the Rover firmware running optimally [21]. The Complete Parameter List section of the documentation is a very helpful reference tool. There is, however, some lack of clarity in the ArduPilot documentation that caused confusion and unexpected behavior when configuring UGV01.

The R/C inputs required troubleshooting for the DX8 to work properly with the Pixhawk. Before adjusting the ArduPilot Parameters, the DX8 controller had to be configured for its signal output to be compatible with the Pixhawk. The Pixhawk does not interpret the DX8 signal correctly with default channel assignments. The “Rx Port Assignments” on the DX8 must be changed to accommodate the Pixhawk such that channel 3 (ELE) is received as throttle, and channel 2 (AIL) is received as roll (steering). To access these settings on the DX8, the following menu items must be selected, beginning on the Function List menu: 1) System Setup, and 2) Channel Assign. Selecting Channel Assign brings the user to the Rx Port Assignments menu where the assignments can be customized to those prescribed in Table 4.2. Selecting NEXT brings the user to the Channel Input Config menu. Here, channels 5-8 can be assigned a specific tactile input. The inputs for channels 1-4 are unchangeable because they are automatically assigned by the DX8 processor.

Table 4.2 - Spektrum DX8 channel assignments for Pixhawk UGV01 configuration

DX8 Channel	*Rx Port Assignment*	*User Input*	*UGV01 Function*
1 (THR)	Elevator	Left Stick (U/D)	N/A
2 (AIL)	Throttle	Right Stick (L/R)	Steering
3 (ELE)	Aileron	Right Stick (U/D)	Throttle
4 (RUD)	Rudder	Left Stick (R/L)	N/A
5 (GER)	Gear	Switch A	Learn Cruise
6 (AUX1)	Aux 1	Switch D	Mode Selector
7 (AUX2)	Aux 2	Switch F	N/A
8 (AUX3)	Aux 3	Switch G	Arm/Disarm

The Pixhawk parameters that control the interpretation of R/C signal from the DX8 are shown in Table 4.3. The full list of parameters relevant to this chapter are found in Appendix

A.1. The values shown for these parameters allow the DX8 and Pixhawk to work well together. The full list of parameters relevant to this chapter are found in Appendix A.1.

Intuitively, the channel assignments are not logical. One would expect the channel assignments on the DX8 in Table 4.2 to line up with the parameters in Table 4.3 (channel 2 would be aileron and channel 3 would be throttle). After testing several channel configuration combinations, it is known that the configurations in Tables 4.2 and 4.3 provide the best DX8 and Pixhawk compatibility. For manual of control of UGV01 with the Pixhawk to be identical to the R/C configuration, PILOT_STEER_TYPE must be set to 0. ArduPilot recommends a value of 2 for skid-steering input rovers, but the DX8 is not the conventional skid-steer controller. SERVO1_FUNCTION and SERVO3_FUNCTION are the parameters that define the steer type of UGV01 as skid-steer. PILOT_STEER_TYPE only defines the R/C input method for manual control.

Table 4.3 - ArduPilot parameters for DX8 input compatibility

Parameter	***Value***	***Function***
RCMAP_PITCH	1	Map pitch to channel 1
RCMAP_ROLL	2	Map roll to channel 2
RCMAP_THROTTLE	3	Map throttle to channel 3
RCMAP_YAW	4	Map yaw to channel 4
PILOT_STEER_TYPE	0	Default single-joystick R/C input
SERVO1_FUNCTION	73	Servo 1 controls the left track
SERVO3_FUNCTION	74	Servo 3 controls the right track

Additional functionality was given to the DX8 transmitter via auxiliary function parameters in ArduPilot. The channel assignments detailed in Table 4.2 show that channels 5-8 were mapped to three-position switches. Switch D (channel 6) was mapped to the mode selector function. On a three-position switch, there are three PWM outputs: 1100 microseconds, 1500 microseconds, and 1900 microseconds. ArduPilot supports the assignment of six modes to six PWM ranges on the mode selector channel. Thus, the low (MODE1), mid (MODE4), and high (MODE6) ranges were assigned as auto, manual, and hold modes on the three-position switch. Switch A (channel 5) was assigned the "Learn Cruise" function, which teaches the Pixhawk the speed it should reach when in auto mode. Switch G (channel 8) was assigned the arm/disarm function.

Table 4.4 - Parameters for auxiliary functions on DX8

Parameter	*Value*	*Function*
MODE_CH	6	Map mode selector to channel 6
MODE1	10	Auto mode assigned to low PWM
MODE4	0	Manual mode assigned to neutral PWM
MODE6	4	Hold mode assigned to high PWM
RC5_OPTION	50	Map Learn Cruise function to channel 5
RC8_OPTION	41	Map arm/disarm function to channel 8

After configuring the DX8 and Pixhawk for basic compatibility and functionality, the system was tuned to achieve optimal performance. First, the positions of sensors on the body of UGV01 were given to ArduPilot. The default position of the GPS and accelerometer/gyroscope is the centroid of the vehicle. Stacking all sensors at the centroid of UGV01 was not possible due to geometric constraints and sensor interference. Thus, the offsets were adjusted via the parameters in Table 4.5.

Table 4.5 - ArduPilot parameters for sensor orientation

Parameter	*Value (meters)*	*Function*
INS_POS1_X,	0.04	X offset for accelerometer/gyro
INS_POS1_Y,	0.05	Y offset for accelerometer/gyro
INS_POS1_Z,	0.045	Z offset for accelerometer/gyro
GPS_POS1_X,	0.185	X offset for GPS/compass
GPS_POS1_Y	0	Y offset for GPS/compass
GPS_POS1_Z	-0.175	Z offset for GPS/compass

ArduPilot requires a one-time calibration of the accelerometer and compass before the motors can be armed. The accelerometer calibration correlated accelerometer readings to different body orientations of UGV01. The compass calibration allowed the Pixhawk to compensate for ferrous metal in the frame of UGV01. As previously noted, there is an external compass in the Pixhawk GPS module and an internal compass on the Pixhawk 4 board. Since the internal compass was surrounded by metal and nearby UGV01's drive motors, its readings were unreliable. Therefore, the external compass was made primary and internal compass disabled via the parameters in Table 4.6.

Table 4.6 - ArduPilot parameters for exclusive use of the external compass

Parameter	*Value*	*Function*
COMPASS_PRIMARY	0	Make first compass primary
COMPASS_USE	1	Enable first (external) compass
COMPASS_USE2	0	Disable second (internal) compass

The ability of the Pixhawk to move UGV01 predictably and accurately to achieve velocity setpoints was crucial. For example, it is best if UGV01 moves in a straight line when commanded to do so. If it veers slightly left or right, the Pixhawk must detect and correct the error, resulting in a non-linear path.

The right-angle drill motors on UGV01 are timed such that a higher speed is achieved in the clockwise direction. As a result of this non-neutral timing, equal and opposite voltages applied to a motor will not produce equal and opposite angular velocities. Since the left and right motors on UGV01 are identical, one must spin clockwise and the other counterclockwise to move both tracks forward. This design causes UGV01 to naturally veer right when equal and opposite voltages are applied to the left and right motors. To counteract the non-neutral timing, and thus the rightward veering, the PWM outputs to the left and right motors were tuned. To equilibrate the forward speeds of the tracks, the left motor had to be slowed by reducing its maximum PWM output from 2000 microseconds to 1880 microseconds. The neutral PWM outputs were kept at 1500 microseconds and the minimum throttle was set to 4% to avoid problems with the dead zone and unequal static and Coulomb friction. The PWM tuning values in Table 4.7 are in microseconds.

Table 4.7 - ArduPilot parameters for tuning motor PWM outputs

Parameter	*Value*	*Function*
SERVO1_MAX	1880	Maximum PWM output for left motor
SERVO1_MIN	1100	Minimum PWM output for left motor
SERVO1_TRIM	1500	Neutral PWM output for left motor
SERVO3_MAX	1950	Maximum PWM output for right motor
SERVO3_MIN	1100	Minimum PWM output for right motor
SERVO3_TRIM	1500	Neutral PWM output for right motor
MOT_THR_MIN	4	Minimum throttle % applied by Pixhawk

Chapter 5
Testing UGV01 with Pixhawk

5.00 Overview

This chapter includes the testing methods and results for the autonomous control of UGV01 by the Pixhawk. The testing facility is also described. A path-marking device was created to track the path of UGV01 throughout its missions.

5.01 Creating a Test Track at Rock Springs Orchard

The designated test facility for the autonomous orchard mower was the Russell E. Larson Agricultural Research Center at Rock Springs. The apple orchard within this research center was named Rock Springs Orchard. Rock Springs Orchard has six blocks of trees, labeled on the map in Figure 5.1.

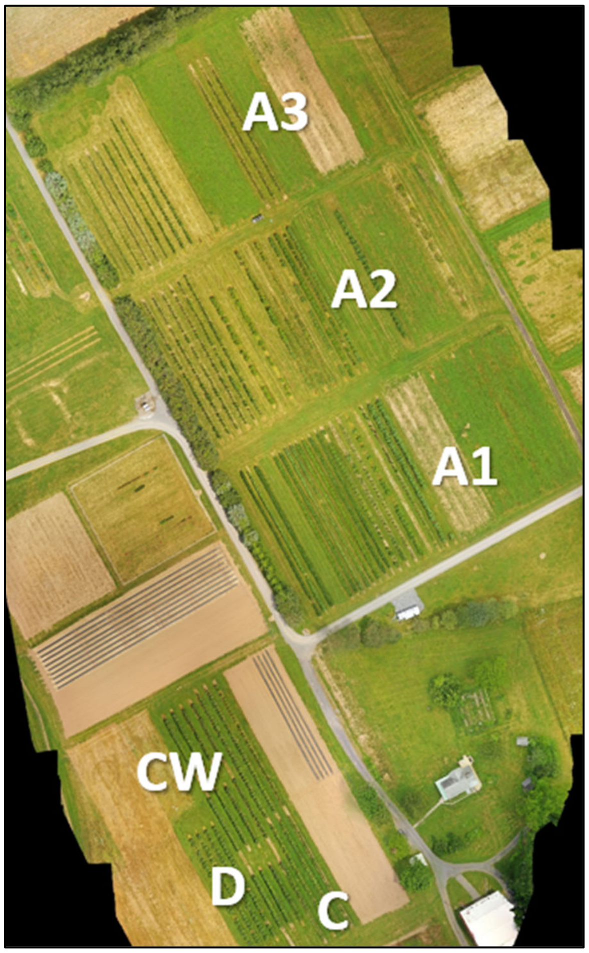


Figure 5.1 - Rock Springs Orchard

A ground control point, or GCP, is a physical landmark with a known latitude, longitude, elevation, and degree of accuracy. Across Rock Springs Orchard, there exist six GCPs with a latitude, longitude, and elevation precision of three centimeters. The locations of the six GCPs were found by Dr. Sean Brennan's Intelligent Vehicles and Systems Group within Penn State's Department of Mechanical Engineering. The Intelligent Vehicles and Systems Group utilized their mapping van equipped with DGPS to find precision coordinates. Each GCP was named based on its location relative to nearby orchard blocks, as shown in Figure 5.2. For example, A1 S is the GCP South of block A1. All GCPs at Rock Springs Orchard are marked with 11.7 in x 11.7 in stone pavers recessed into the sod.

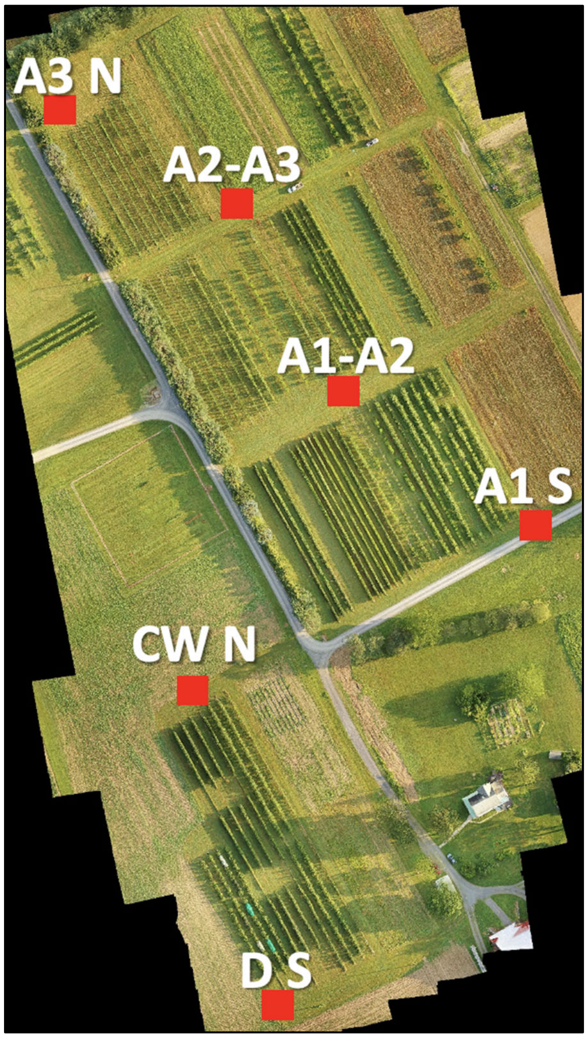


Figure 5.2 – 6 Ground Control Points (GCPs) at Rock Springs Orchard

In order to test the accuracy and precision of the Pixhawk on UGV01, consecutive GCPs were needed as waypoints. Additionally, the paths between the waypoints had to be obstacle-free. As shown in Figure 5.2, no straight-line paths between existing GCPs were obstacle-free. Therefore, new secondary GCPs were needed to form an obstacle-free test track for UGV01. The aisle between blocks A1 and A2 was selected as the testing grounds due to its large width.

The test track was designed to mimic 2 rows of apple trees, the width between the rows being of primary concern. While the width of the orchard rows varied, the approximate average width was found to be 10 ft. Most rows were over 200 ft long, but it was not necessary to make the test track full-length. A shorter track length was desirable for repeated tests because each test would discharge less energy from UGV01's battery. The test track had to be positioned sufficiently far from the west end of the A1-A2 aisle to prevent interference from large pine trees. Not only would these trees act as ground obstacles, but satellite reception near the trees would be diminished.

The trees at Rock Springs Orchard grow on trellises, which are structural posts and wires that run along the length of a row. At the ends of some rows, the trellises had support posts and wires extending beyond the last tree. To incorporate this obstacle into the test track, UGV01 would have to overshoot each GCP by 15 feet, mimicking the avoidance of trellis brace posts. Based on these parameters, four additional stone pavers were recessed to create two mock tree rows, as shown in Figure 5.3. The rows are 10 feet apart, 130 feet long, and the west end of the test track is 30 feet from the large pine trees, as shown in Figure 5.4.

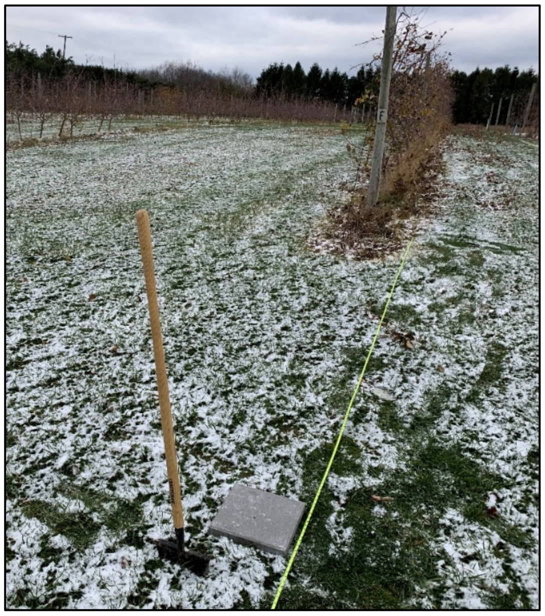


Figure 5.3 – Recessing a stone paver to establish a test track GCP

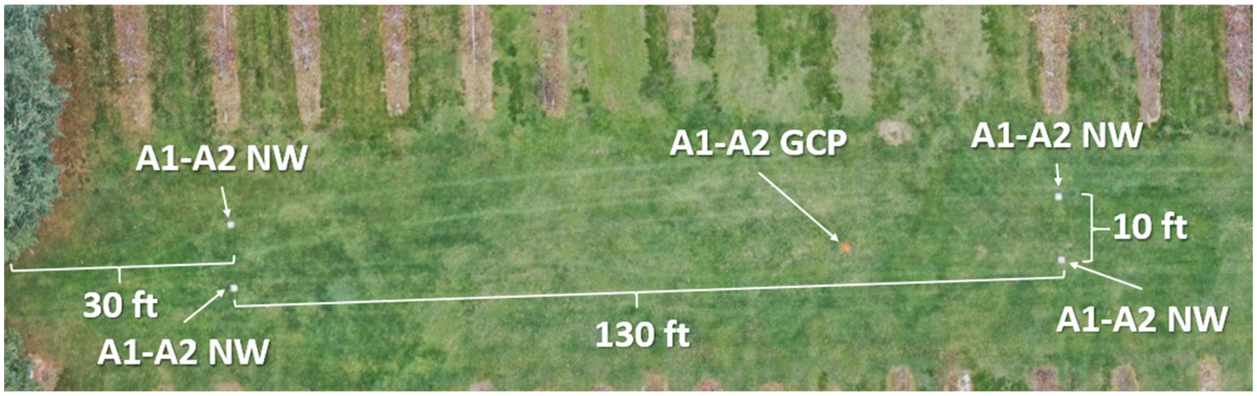


Figure 5.4 - Completed test track in aisle A1-A2 composed of 4 GCPs

To establish each test track paver as a new GCP for Rock Springs Orchard, the coordinates of each paver had to be found to an acceptable degree of accuracy. This was accomplished with photogrammetry using an orthomosaic of Rock Springs Orchard. A UAV outfitted with a high-resolution camera swept across the entire orchard capturing images at a fixed altitude. These images were processed and stitched together to create an undistorted, uniformly-scaled, high-resolution image of orchard. Figure 5.1 and Figure 5.2 are orthomosaics created from drone imagery. Since an orthomosaic of Rock Springs Orchard includes the six original GCPs, the map can be further processed to create a map that is latitudinally and longitudinally calibrated. More information on calibrated map creation is found in Section 7.04. Photogrammetry was then performed with the calibrated map to digitally find the locations of the four test-track secondary GCPs. The accuracy of the secondary locations is estimated to be 14 centimeters. The GCPs were named based on their location within aisle A1-A2. For example, A1-A2 NW is the GCP at the north-west corner of the test track. The GCPs are labeled in Figure 5.4.

5.02 UGV01 Ground-Marking Device

One challenge of testing UGV01 with a basic Pixhawk configuration was quantifying its performance. A basic evaluation of performance entails comparing UGV01's true location with its desired location, namely at waypoints. UGV01 will act based on its perceived location, but the inaccuracies of its GPS/GLONASS L1 antenna cause perceived location to deviate from the true location. The difficulty in comparing true location and desired location is recording the true location of UGV01 throughout a mission. The devised solution was a ground-marking device that traced the center of UGV01 throughout the mission.

The ground markings had to be clearly distinguishable but temporary so consecutive tests could be performed in the same location. In order to protect the health of Rock Springs Orchard, the marking compound also had to be non-toxic. The first marking compound tested was white sand, due to its reliable and smooth flow rate. The test revealed that pure white sand does not create a distinguishable ground mark. Rather, the sand falls past the grass, hiding any sand deposited to the area. To improve the distinction of the marking compound, all-purpose flour was mixed with sand. With this mixture, sand acted as a steady flow solvent with flour as a distinct marker. The amount of flour in the mixture had to be limited due to its tendency to clump and block flow at the aperture. The ideal sand-to-flour mixture ratio was found to be 3:1.

After determining the ideal marking compound, a dispensing device had to be added onto UGV01. The device used a hopper to hold the marking compound, a funneling shape that leads to an aperture, and a mounting bracket. The size of the aperture had to be chosen to achieve the correct flow rate. Based on research from the University of Buenos Aires in Argentina, it is known that the flow rate of sand through an aperture is constant and depends only on the area of the orifice [22]. The flow rate had to be high enough to mark the ground clearly, but low enough to be efficient with the use of the supply in the hopper.

A test was performed to find the baseline flow rate of the marking compound. The bottom of an empty 2-liter soda bottle was removed and a 0.5-inch diameter aperture was bored into the lid. Five hundred (500) milliliters of the marking compound were added to the bottle and then flowed onto the ground as the bottle was horizontally translated over grass at a speed similar to that of UGV01. Five hundred (500) milliliters of the mixture were able to create a line ~100 feet long. While dispensing the compound, the relatively small aperture in the flat lid caused some instances of flow stoppage. To eliminate this problem, a funneled aperture replaced the bored cap as shown in Figure 5.5. The threaded cap from the bottle was glued into a funnel with a 3/8-inch diameter spout. The hole in the cap was enlarged to prevent flow blockage.

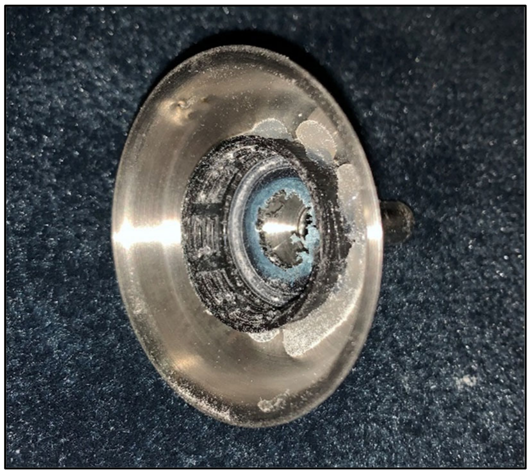


Figure 5.5 - Funnel dispenser for marking compound hopper

To fasten the hopper to UGV01, a 6x3x1/8-inch steel plate was bent into an L bracket and bolted to the aft of UGV01 as shown in Figure 5.6. A 5/16-inch hole was bored into the bracket to snuggly fit the stainless-steel funnel. The hole was offset far enough from the body to accommodate the size of the hopper.

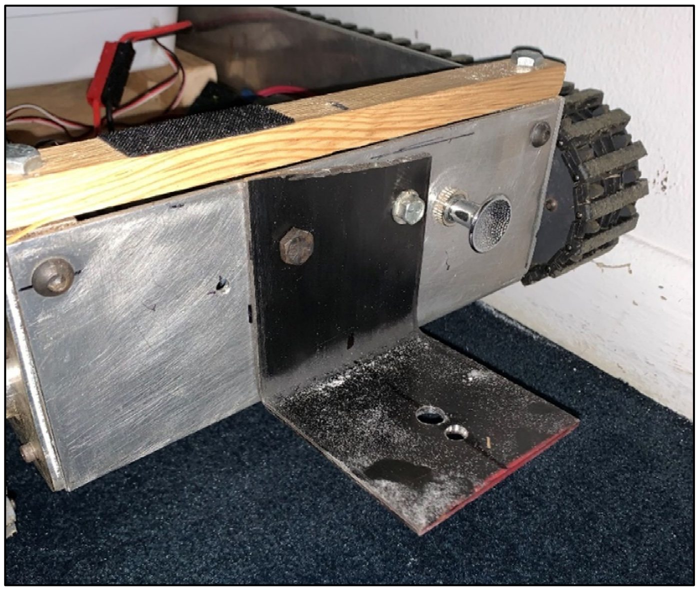


Figure 5.6 - L bracket for marking device mounted to the aft of UGV01

Before field testing the ground-marking device on UGV01, the flow rate of the device was determined. Five hundred (500) milliliters of the compound flowed through the funnel for about 60 seconds, thus the flow rate is 8.33 milliliters per second. Knowing the cruise speed of UGV01 is about 3.8 feet per second and the length of the test track is 130 feet, it takes UGV01 about 34 seconds to complete one pass on the test track. Therefore, approximately 283 milliliters of marking compound are needed for every pass on the test track. To verify the functionality of the ground-marking device mounted to UGV01, marking compound was added to the hopper and UGV01 was driven manually in a grassy area. The path of UGV01 was clearly marked by the device, as shown in Figure 5.7, and the markings were easily cleared from the grass.

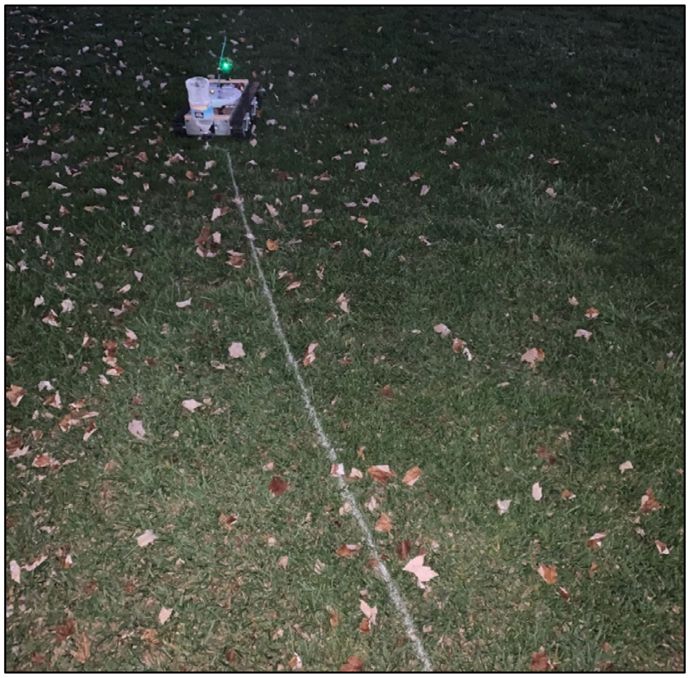


Figure 5.7 - Testing functionality of UGV01 with ground marking device

5.03 Evaluating UGV01 Performance on Test Track

After equipping UGV01 with the ground-marking device, it was brought to the test track at Rock Springs Orchard to evaluate its autonomous performance. In Mission Planner, a mission was devised so that UGV01 would pass over three secondary GCPs in the following order: A1-A2 SW, A1-A2, A1-A2 NE. The last waypoint of the mission was set arbitrarily so that UGV01 would be in motion over the three secondary GCPs. In other words, the secondary GCPs were made dynamic waypoints. Mission visualization provided by Mission Planner is shown in Figure 5.8. It displays all waypoints, including a "Home Position" waypoint. This waypoint is not used by an UGV01 during missions. Home position is further discussed in section 8.04.

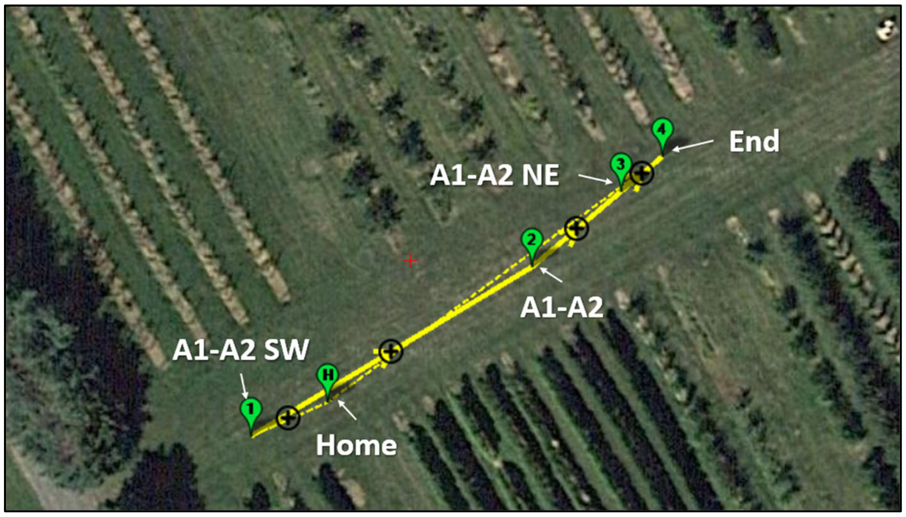


Figure 5.8 - First test track mission displayed in Mission Planner

The mission was uploaded to the Pixhawk and ground-marking compound was filled into the hopper on UGV01. The vehicle was set several feet behind A1-A2 SW with its heading pointed toward the GCP. The Pixhawk was switched into auto mode and UGV01 completed the mission. The procedure was then repeated to document the performance of UGV01 during two independent missions. The Pixhawk was imprecise (inconsistent performance) but achieved a decent level of accuracy (lower deviation) during the second run.

Table 5.1 - Deviation of UGV01 during test track missions

GCP	*Run 1 Deviation [in]*	*Run 2 Deviation [in]*	*Average Deviation [in]*
A1-A2 SW	65	10	37.5
A1-A2	49	1	25
A1-A2 NE	59	7	33

Figure 5.9 through Figure 5.11 show the ability of the ground-marking device to quantify the performance of UGV01 during an autonomous mission. Figure 5.9 and Figure 5.10 have white arrows overlaying the ground marking to increase clarity. After the first run, the trace of

UGV01 at each GCP was documented and then the line was swept away. Figure 5.12 shows a longer trace made by the ground-marking device, revealing the non-linear path UGV01 took between the two waypoints.

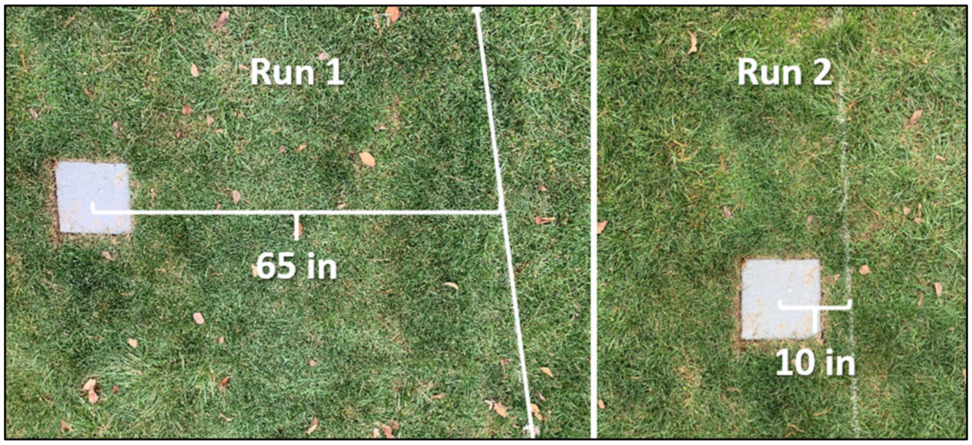


Figure 5.9 - A1-A2 SW

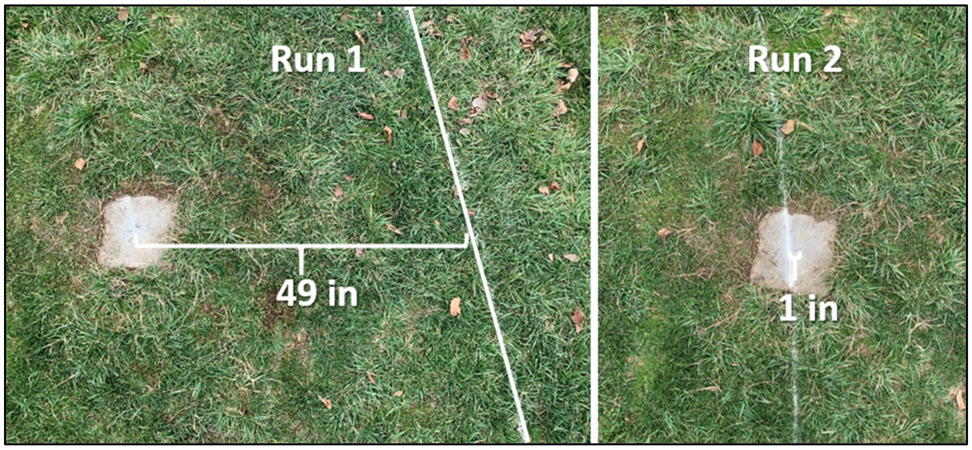


Figure 5.10 - A1-A2

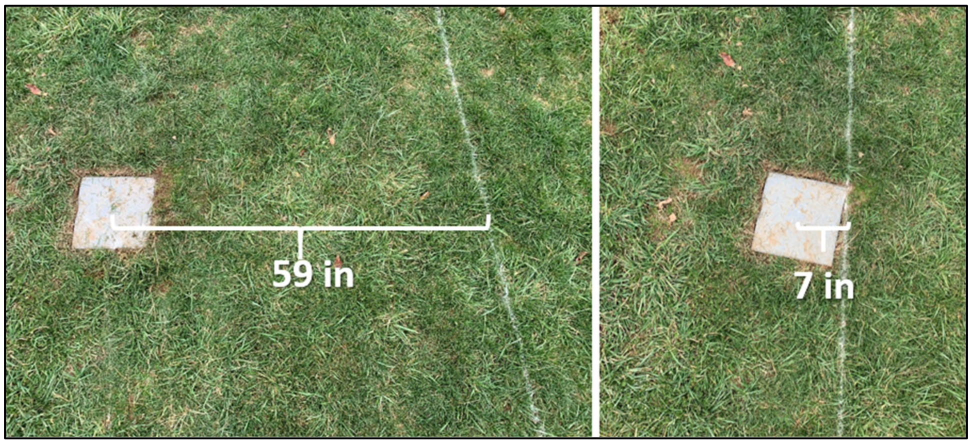


Figure 5.11 - A1-A2 NE

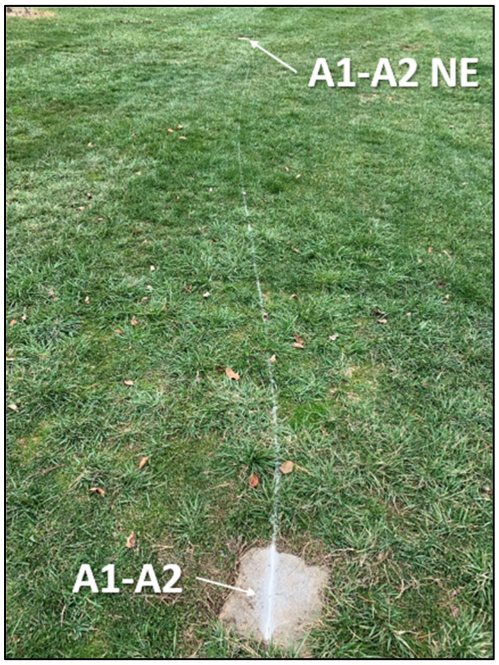


Figure 5.12 - Trace of UGV01 path from A1-A2 to A1-A2 NE during run 2

The performance of UGV01 with the basic Pixhawk L1 GPS/GLONASS antenna was not precise enough for navigation through an orchard. The largest deviation of UGV01—65 inches or 5.41 feet—makes this system incompatible with orchard row navigation. The average width of an orchard row is about 10 feet and the narrowest rows can be about 8 feet wide. The cutting width of the Cub Cad RZT-S Zero is about 42 inches. This leaves 39 inches between tree trunks and either side of the mowing deck. Protruding branches make row width narrower, thus in practice the allowable deviation is less than 10 inches. For the Pixhawk system to be capable of safely navigating between tree rows at Rock Springs Orchard, the precision and accuracy of its satellite localization system had to be improved.

Chapter 6
Establishing an RTK System with Pixhawk

6.00 Overview

This chapter describes the methods used to establish and integrate RTK GNSS into the Pixhawk on UGV01. The objective of integrating RTK GNSS was to increase the precision and accuracy of the location information provided to Pixhawk, consequently minimizing UGV01's deviations from the mission path.

6.01 Hardware Configuration

To provide the Pixhawk with high-precision positioning data, an RTK-capable receiver must be connected to the Pixhawk and correction signals must be supplied to the receiver. The u-blox ZED-F9P L1/L2 receiver on the SparkFun GPS-RTK-SMA board was selected as the RTK receiver. In some geographic locations, corrections signals are publicly-available from real-time networks. To utilize the correction signals, a real-time station must be within 10 kilometers of the RTK receiver. Since closest real-time station to Rock Springs Orchard is approximately 77 kilometers away, a dedicated real-time corrections source had to be established. Fortunately, the ZED-F9P module can be configured as either a base (i.e. real-time station) or rover.

Two SparkFun ZED-F9P boards were acquired to establish the RTK system. Each board also required a u-blox L1/L2 GNSS antenna to receive signals from GPS, GLONASS, Galileo and BeiDou satellites. The base communicates with the rover via a telemetry radio pair. As noted in Chapter 2, dual-band GNSS receivers provide a higher level of precision than GPS or single-band GNSS receivers. The ZED-F9P RTK modules are capable of one-centimeter horizontal precision [23]. Nathan Seidle from SparkFun has published some helpful documentation for configuring the ZED-F9P boards [24–27].

The base hardware consists of the ZED-F9P board, L1/L2 GNSS antenna, and telemetry radio. The female SMA connector on the GNSS antenna simply connects to its male counterpart on the RTK board. Correction data sent from the base follows the common messaging protocol set for communication between base stations and rovers. This protocol was established by the Radio Technical Commission for Maritime Services, hence the corrections signals are referred to as RTCM. RTCM is sent from the ZED-F9P board via the RTCM pins, shown in Figure 6.1. These pins are connected to the “correction UART” chip, labeled UART2. A UART, or Universal Asynchronous Receiver Transmitter, is a device that allows for simultaneous sending and receiving of serial data. Serial communication is the transmission of data over a single channel, one bit at a time. Two critical pins on a UART interface are Tx and Rx. Data is transmitted out of a UART device from Tx pin and received into the Rx pin.

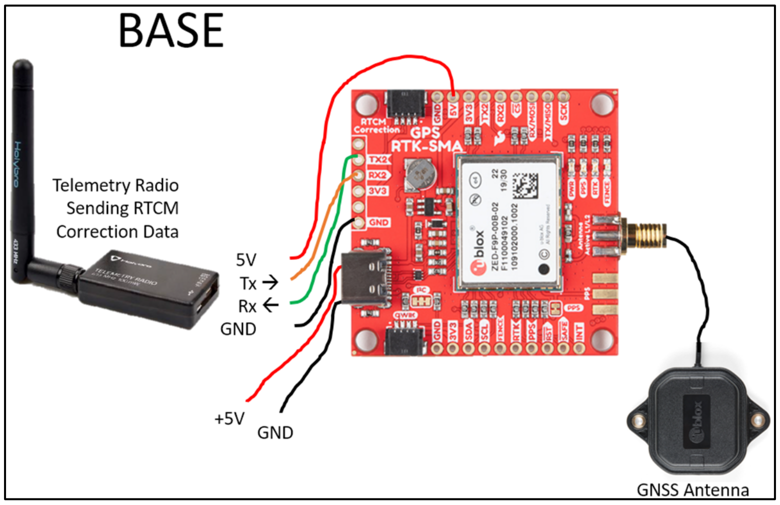


Figure 6.1 - Hardware configuration of the RTK base station

To wirelessly transmit the serial data leaving the UART2 port, a telemetry radio is wired to the 5V power source, ground, Tx2, and Rx2 pins, as shown in Figure 6.1. Logically, the sending Tx pin of the telemetry radio is connected to the receiving Rx pin of the RTK board, and vice versa. It should be noted that the orange line connecting the radio Tx pin and RTCM Rx2 pin is not transmitting any data. Corrections signals are only sent out from the board, not received. Table 6.1 summarizes the wiring connections made to the RTK base module.

Table 6.1 - RTK base wiring connections

Board Pinout	*External Pinout*	*Function*
TX2	Radio Rx	Send RTCM
RX2	Radio Rx	None
GND	Radio GND	Radio ground
5V	Radio 5V	Radio power
USB-C port	5V USB source	Board power
SMA connector	GNSS antenna	Receive satellite data

The board is powered by a 12 V lead-acid battery with a 5V voltage converter via the USB-C port. The board is housed within a waterproof electronics box and mounted to the top of a tripod. Protruding from the box are the telemetry radio antenna, power lead, and GNSS antenna SMA cable, as shown in Figure 6.2.

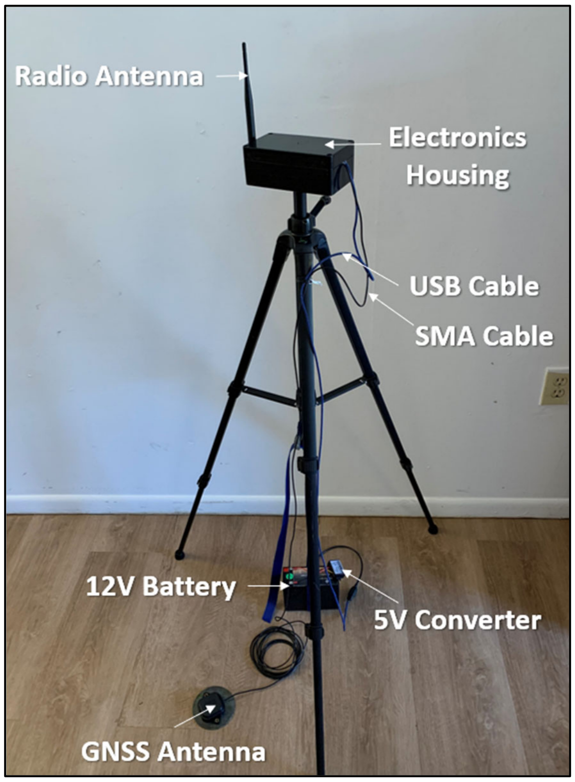


Figure 6.2 - RTK base module assembled with critical components

The rover RTK hardware configuration is similar to the base and is shown in Figure 6.3. UART2 is wired to a telemetry radio in an identical fashion. In the case of the rover, however, the green line connecting the RTCM Tx2 pin to the radio Rx pin is not transmitting data. RTCM data is only received by the board. The rover RTK board supplies high-accuracy location data to the Pixhawk via the UART1 Tx pin. The standard messaging protocol used for the satellite location data is defined by the National Marine Electronics Association (NMEA). The rover RTK board uses NMEA messages to communicate location data to the Pixhawk. The Serial4 port on the Pixhawk was used to receive these NMEA messages. Justification for the use of this port is found in Section 6.04. The Serial4 port accepts a six wire JST-GH type cable. Note that all Pixhawk ports accept JST-GH type cables. The Pixhawk supplies 5V power to the rover RTK board via the Serial4 port. The wiring connections made for the RTK rover module are summarized in Table 6.2.

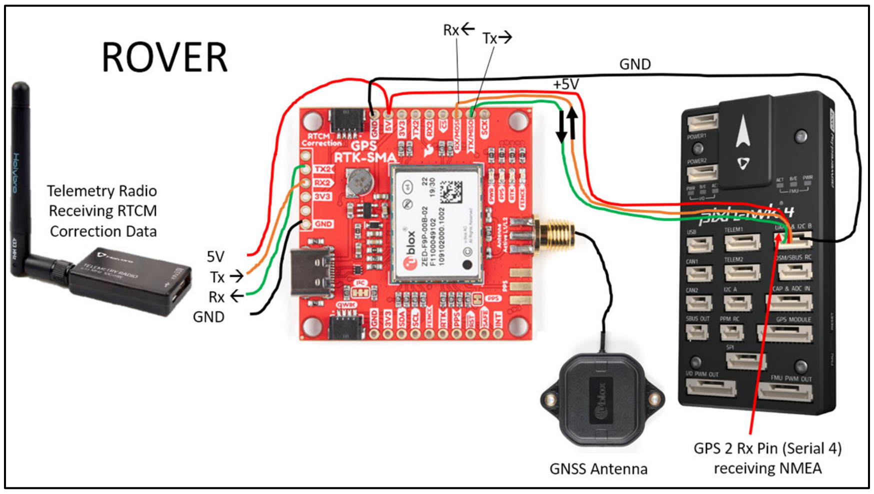

ROVER
Rx←
Tx→
GND
+5V
Telemetry Radio Receiving RTCM Correction Data
5V
Tx →
Rx ←
GND
GNSS Antenna
GPS 2 Rx Pin (Serial 4) receiving NMEA

Figure 6.3 - Hardware configuration of RTK rover module with the Pixhawk

Table 6.2 - RTK rover wiring connections

Board Pinout	***External Pinout***	***Function***
TX2	Radio Rx	None
RX2	Radio Rx	Receive RTCM
GND	Radio GND	Radio ground
5V	Radio 5V	Radio power
5V	Pixhawk Serial4 5V	Board power
RX/MOSI	Pixhawk Serial4 Tx	None
TX/MISO	Pixhawk Serial4 Rx	Send NMEA
GND	Pixhawk Serial4 GND	Board ground
SMA connector	GNSS antenna	Receive satellite data

To accommodate the new RTK system hardware on UGV01, a water-proof electronics box was installed on the vehicle. Exiting the box is the Pixhawk telemetry radio antenna, RTCM radio antenna, DSMX satellite receiver wires, motor signal wires, PMB power leads, GNSS antenna SMA cable, and USB-C cable. The USB-C cable is used for connecting a PC to the RTK board for troubleshooting and data-logging. The hardware components contained within the UGV01 electronics box are shown in Figure 6.4.

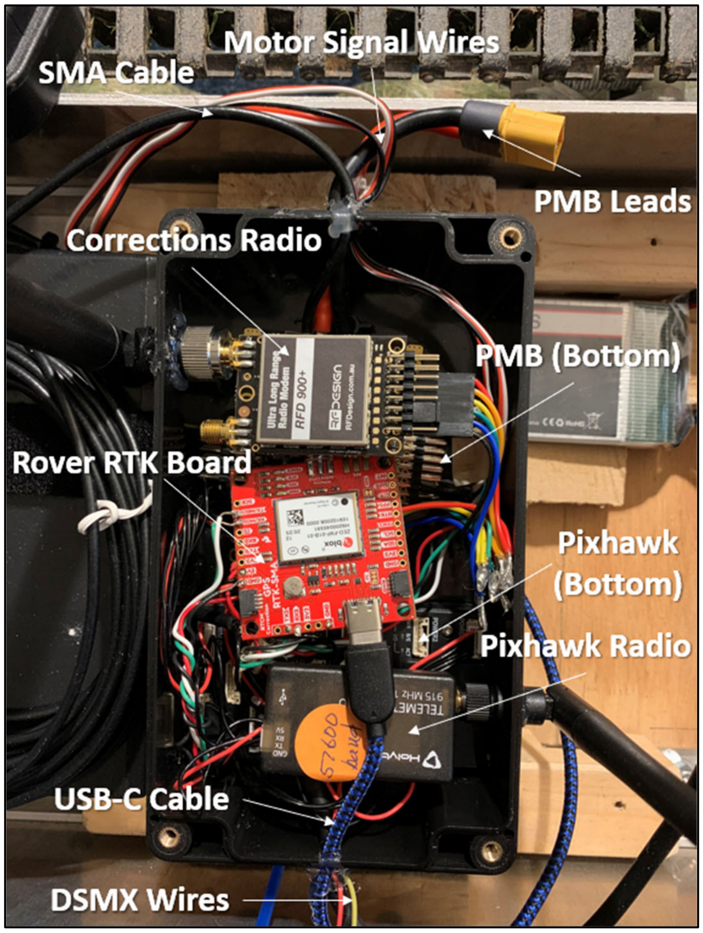


Figure 6.4 – Rover RTK hardware within protective housing on UGV01

Demanding high-accuracy autonomous performance from the Pixhawk necessitated an improvement of the UGV01 prototype. Namely, the compass needed a ferrous-free mount away from electronics. A pedestal with a carbon rod was used to mount the compass. Also, the electronics needed a sturdier more protective housing. As shown in Figure 6.5, the arrangement of hardware within the vehicle was redesigned and a wooden deck was installed on the aft.

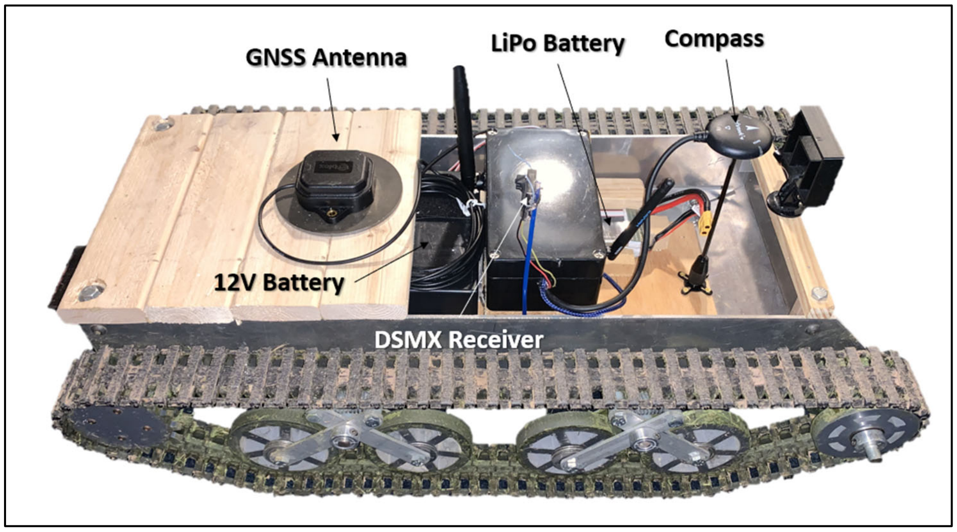


Figure 6.5 - Updated design of UGV01 with Pixhawk and RTK system

This deck protected the motors, Sabertooth motor controller, battery, and wire terminals. The deck also provided a surface to mount the GNSS antenna. The GNSS antenna used in the RTK system requires a 4-inch diameter ground plane (a circular steel plate) for optimal performance. The grounding plane is mounted to the deck and the antenna magnetically adheres to this grounding plane as shown in Figure 6.6.

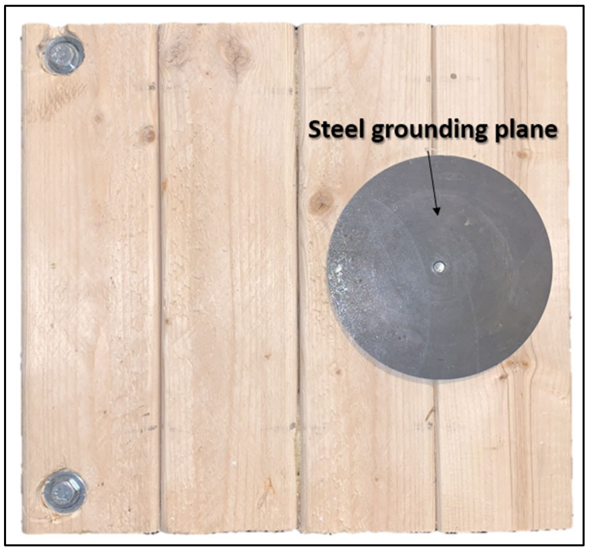


Figure 6.6 - UGV01 aft deck with GNSS antenna grounding plane

6.02 Cost Consideration

Integrating the RTK system with the Pixhawk is necessary to achieve acceptable accuracy. However, it is also critical to keep cost in mind. The cost of commercial RTK GPS systems is traditionally a barrier to entry when it comes to precision farming. One modern agriculture guidance system, FieldBee, offers a complete RTK L1 GPS system for $1,850 [28]. Even at this high price tag, the GNSS antennas within the rover and base modules only receive one frequency band (L1 or L2). Therefore, the ZED-F9P RTK system receives more satellite signals than the FieldBee, making its location solutions more consistent and reliable. The cost of the ZED-F9P RTK base and rover system is broken down in Table 6.3.

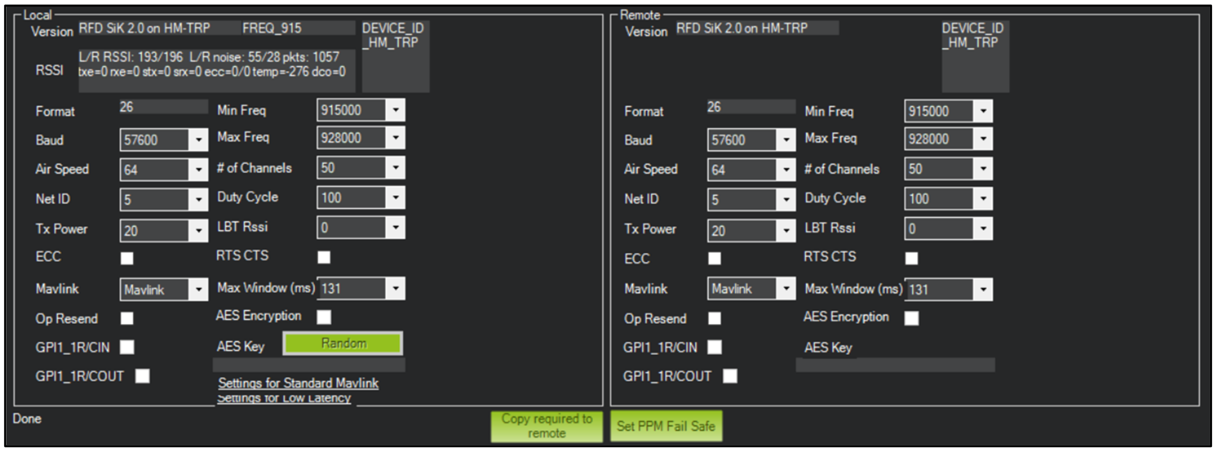
The total cost of the system is approximately $850, which is $1000 less than the FieldBee commercial RTK system. The specifications of the ZED-F9P RTK board is indicative of the cost-effectiveness of current satellite localization technology. The higher price tag of consumer products will fall as low-cost, high precision localization technology becomes more widely available. With affordable high-precision satellite localization technology, cost-effective mission-based autonomy can be achieved for orchard vehicles.

Table 6.3 – Cost sheet of Pixhawk RTK GPS system

Component	*Quantity*	*Unit Cost*
Sparkfun RTK Board	2	$219.95
Long Range Radio Modem	2	$109.50
Long Range Antenna	2	$6.05
Electronics Box	2	$8.50
12 V Battery	1	$24.50
5V Converter	1	$9.86
L1/L2 GNSS Antenna	2	$64.95
Total Cost		$852.25

6.02 Configuring Two Telemetry Radio Pairs

The RTK GPS system requires a radio connection from the base to the rover for transmission of RTCM corrections. This adds a second pair of telemetry radios to the Pixhawk system, as shown in Figure 6.4: one radio pair for Pixhawk to Mission Planner communication and one pair for RTCM. With factory default configurations, two independent telemetry radio connections cannot be made without causing major inference and miscommunication of information. To reconfigure the radios, SiK Radio configuration software was used. SiK radios are characterized by their lightweight firmware and hardware [29]. Within Mission Planner under Setup, then Optional Hardware, a SiK radio configuration tool can be found. To establish two independent radio connections, the Net IDs set on each pair of radios must be unique. For the Pixhawk to Mission Planner radio pair, a Net ID of 5 was set. For the RTCM radio pair, a Net ID of 105 was set. While all other settings can be left to their default values, it is important to ensure the baud rate for each radio pair is identical. The baud rate is the speed of the serial connection in bits per second. For either serial device to correctly interpret serial data, it must know what speed at which the bits are sent. The baud rate for all four radios was left at 57600 bits per second. An example of the Mission Planner SiK radio configuration tool is shown in Figure 6.7.



Local

Version: RFD SiK 2.0 on HM-TRP FREQ_915 DEVICE_ID_HM_TRP

RSSI: L/R RSSI: 193/196 L/R noise: 55/28 pkts: 1057 txe=0 rxe=0 stx=0 srx=0 ecc=0/0 temp=-276 dco=0

Format	26	Min Freq	915000
Baud	57600	Max Freq	928000
Air Speed	64	# of Channels	50
Net ID	5	Duty Cycle	100
Tx Power	20	LBT Rssi	0
ECC	☐	RTS CTS	☐
Mavlink	Mavlink	Max Window (ms)	131
Op Resend	☐	AES Encryption	☐
GPI1_1R/CIN	☐	AES Key	
GPI1_1R/COUT	☐		

Remote

Version: RFD SiK 2.0 on HM-TRP DEVICE_ID_HM_TRP

Format	26	Min Freq	915000
Baud	57600	Max Freq	928000
Air Speed	64	# of Channels	50
Net ID	5	Duty Cycle	100
Tx Power	20	LBT Rssi	0
ECC	☐	RTS CTS	☐
Mavlink	Mavlink	Max Window (ms)	131
Op Resend	☐	AES Encryption	☐
GPI1_1R/CIN	☐	AES Key	
GPI1_1R/COUT	☐		

Figure 6.7 - SiK radio configuration tool within Mission Planner

An important consideration for the RTCM telemetry radio pair is signal strength throughout the orchard. The performance of UGV01 depends on the accuracy of the rover RTK location solution, which relies upon the RTCM corrections. If signal is poor, performance will falter. The first set of radios used for corrections were Holybro 915-megahertz 100-milliwatt radios. This Holybro radio can be seen clearly in Figure 6.1 and Figure 6.3.

To evaluate signal strength throughout the orchard, the SiK radio software was used to capture the local received signal strength indicator (RSSI) by the rover radio. Shown in Figure 6.7, the configuration tool provides the local and remote RSSI's proceeding "L/R RSSI." The maximum RSSI for the SiK radios was found to be 220. With the RTK base corrections radio set at the A1 South GCP, the RSSI of the rover radio at all GCPs was documented. Percentage RSSI is calculated by dividing the RSSI by the maximum RSSI of 220.

As shown in Table 6.4, the maximum signal strength reported by the rover corrections radio was 32%. In light of the poor signal strength achieved by the Holybro radios, the corrections radios were upgraded to a pair of RFD900 915-megahertz radios. When supplied with 5V, the RFD900 has a power output of 790 milliwatts, compared to the 100-milliwatt output of the Holybro radio. The RFD900 corrections radio is pictured in Figure 6.4. To properly wire the RFD900 radio to the RTK boards, the RFD900 datasheet was referenced. Figure 6.8 shows the pinouts of the RFD900.

Table 6.4 - RSSI of Holybro corrections radio throughout the orchard

Rover Location	*Rover RSSI*	*Rover % RSSI*
A1-A2	65	30%
A2-A3	53	24%
A3 N	41	19%
CW N	71	32%
D S	45	20%

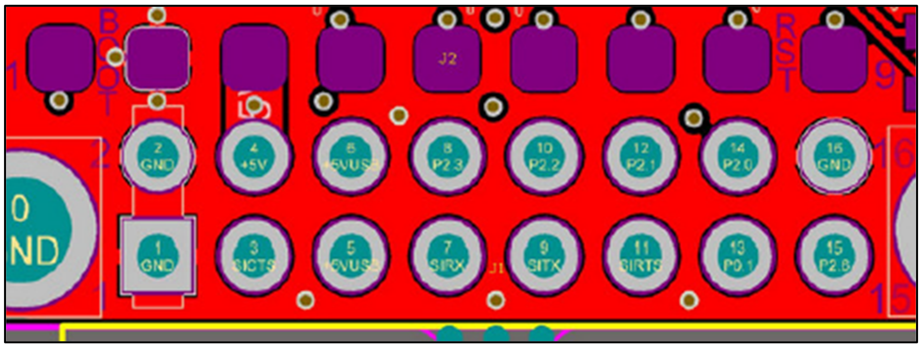


Figure 6.8 - RFD900 radio modem pin layout [30]

Using the SiK radio software, the RFD900 radio settings were configured to match the settings of the Holybro RTCM pair: NETID and baud rate set to 105 and 57600, respectively. The same procedure was repeated to test the signal strength throughout the orchard and is summarized in Table 6.5. Percentage RSSI was improved by at least 25% at each location, with the lowest signal strength being 56% at the D South GCP.

Table 6.5 - Comparison of Holybro and RFD900 radio signal strengths

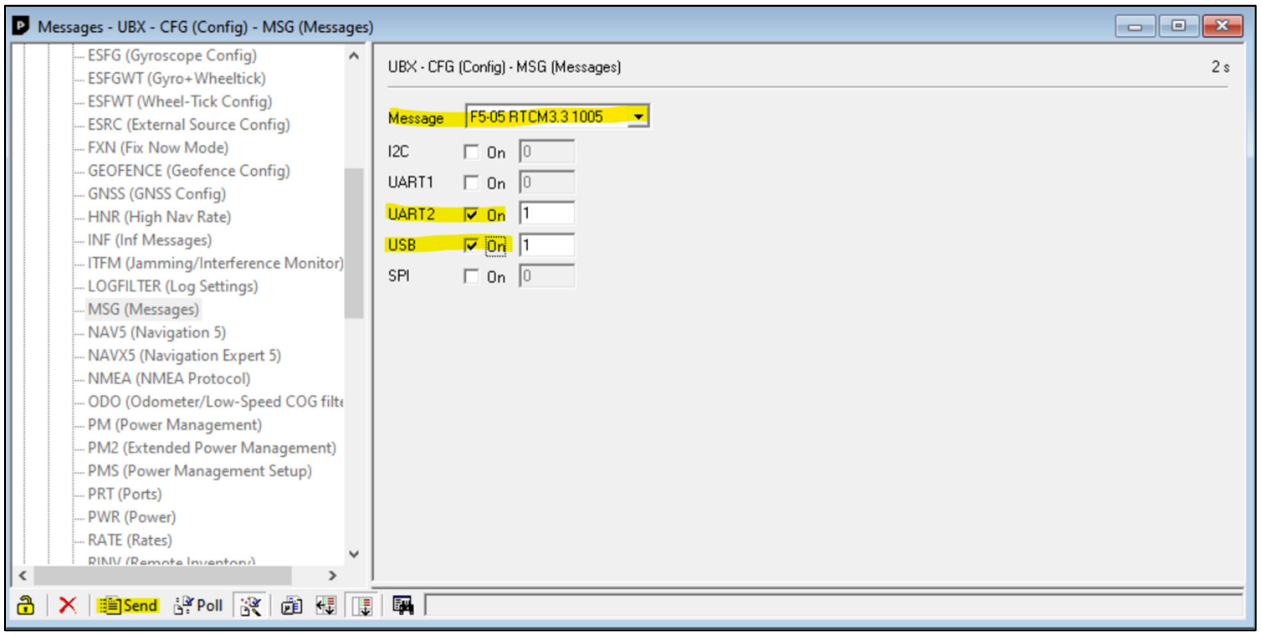
	Holybro Radio Base		***RFD 900 Radio Base***		
Location	**Rover RSSI**	**Rover % RSSI**	**Rover RSSI**	**Rover % RSSI**	***% Gained***
A1-A2	65	30%	143	65%	35%
A2-A3	53	24%	131	60%	35%
A3 N	41	19%	129	59%	40%
CW N	71	32%	126	57%	25%
D S	45	20%	124	56%	36%

6.03 Software Configuration of ZED-F9P RTK Boards

After wiring the ZED-F9P boards as a base and rover pair, software settings must be adjusted to assign each board its role as either a base or rover. To configure the software on each board, the USB-C port is used to connect to a windows PC running u-blox u-center, the manufacturer's configuration and evaluation software. The documentation published by Nathan Seidle from SparkFun is helpful for configuring ZED-F9P boards through u-center [26,27]. U-center can be downloaded from the u-blox website [31].

First, the firmware for both RTK modules was updated by downloading the latest ZED-F9P firmware (version 1.13) from u-blox. Within u-center, the Firmware Update utility is found under Tools.

The base module was configured as a fixed-base reference station. To achieve this function, the board was given the location of its GNSS antenna and the UART2 port was set to send out RTCM messages. To configure the message settings within u-center, one must select View, then Messages View, UBX, CFG, and finally MSG. After selecting the desired message type, transmission port, and transmission frequency, one must use Send to apply the setting to the connected board. Figure 6.9 shows an example of RTCM message configuration, highlighting the relevant fields. USB was selected as an additional RTCM transmission port for troubleshooting purposes. The number fields to the right of the checkbox are the periods of the message cycle for RTCM. For example, a value of 1 is one message every second and a value of 5 is one message every five seconds.


Messages - UBX - CFG (Config) - MSG (Messages)
ESFG (Gyroscope Config)
ESFGWT (Gyro+Wheeltick)
ESFWT (Wheel-Tick Config)
ESRC (External Source Config)
FXN (Fix Now Mode)
GEOFENCE (Geofence Config)
GNSS (GNSS Config)
HNR (High Nav Rate)
INF (Inf Messages)
ITFM (Jamming/Interference Monitor)
LOGFILTER (Log Settings)
MSG (Messages)
NAV5 (Navigation 5)
NAVX5 (Navigation Expert 5)
NMEA (NMEA Protocol)
ODO (Odometer/Low-Speed COG filt
PM (Power Management)
PM2 (Extended Power Management)
PMS (Power Management Setup)
PRT (Ports)
PWR (Power)
RATE (Rates)
Send
Poll
2 s

UBX - CFG (Config) - MSG (Messages)

Message	F5-05 RTCM3.3 1005	
I2C	☐ On	0
UART1	☐ On	0
UART2	✓ On	1
USB	✓ On	1
SPI	☐ On	0

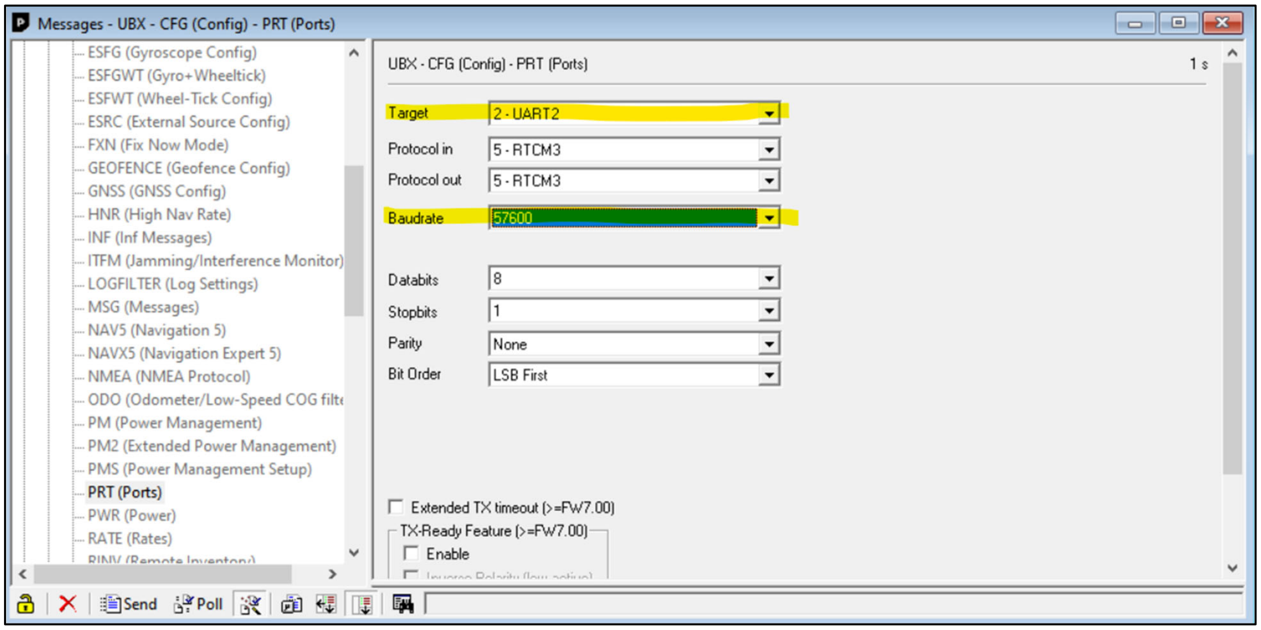
Figure 6.9 - u-center configuration window for base RTCM messages

Several RTCM message types must be sent out from the base RTK module. Each message type contains specific information. The message types sent from the base RTK module are summarized in Table 6.6. Message contents of the RTCM messages were sourced from an article on the SNIP knowledge base [32].

Table 6.6 - RTCM message types sent from the RTK base module

RTCM3.3 Message Type	*Period [s]*	*Message Contents*
1005	1	Location of stationary antenna, quarter phase alignment details
1074	1	Type 4 Multiple Signal Message (MSM) for GPS (USA)
1084	1	Type 3 MSM4 for GLONASS (Russia)
1094	1	Type 4 MSM for Galileo (Europe)
1124	1	Type 4 MSM for BeiDou (China)
1230	5	GLONASS L1, L2 Code-Phase Biases

For proper radio transmission of the RTCM messages sent out of the UART2 Tx, the baud rate of the UART2 port must match the baud rate of the telemetry radio. The long-distance corrections radios' baud rate was set to 57600 bits per second, a sufficient speed for RTCM. The protocol and baud rate for each port on the ZED-F9P board can be set in u-center. To access the settings one must select View, Messages View, UBX, CFG, and then PRT. The only port that must be configured for the base module is UART2. The configuration settings for this port on the RTK base station are shown in Figure 6.10. Note that the port only sends RTCM messages, so the receiving protocol is irrelevant.


Messages - UBX - CFG (Config) - PRT (Ports)
ESFG (Gyroscope Config)
ESFGWT (Gyro+Wheeltick)
ESFWT (Wheel-Tick Config)
ESRC (External Source Config)
FXN (Fix Now Mode)
GEOFENCE (Geofence Config)
GNSS (GNSS Config)
HNR (High Nav Rate)
INF (Inf Messages)
ITFM (Jamming/Interference Monitor)
LOGFILTER (Log Settings)
MSG (Messages)
NAV5 (Navigation 5)
NAVX5 (Navigation Expert 5)
NMEA (NMEA Protocol)
ODO (Odometer/Low-Speed COG filte
PM (Power Management)
PM2 (Extended Power Management)
PMS (Power Management Setup)
PRT (Ports)
PWR (Power)
RATE (Rates)
UBX - CFG (Config) - PRT (Ports)
1 s
Send
Poll

Target: 2 - UART2
Protocol in: 5 - RTCM3
Protocol out: 5 - RTCM3
Baudrate: 57600
Databits: 8
Stopbits: 1
Parity: None
Bit Order: LSB First
☐ Extended TX timeout (>=FW7.00)
TX-Ready Feature (>=FW7.00)
☐ Enable

Figure 6.10 - Base RTK module port configuration in u-center

The base module is configured as a fixed-base station, meaning the location of its GNSS antenna would remain the same each time the board is powered on. To set the known location of the GNSS antenna for the RTK base station, one must select View, Messages View, UBX, CFG, and then TMODE3. The location of the RTK base station at Rock Springs Orchard is the A1

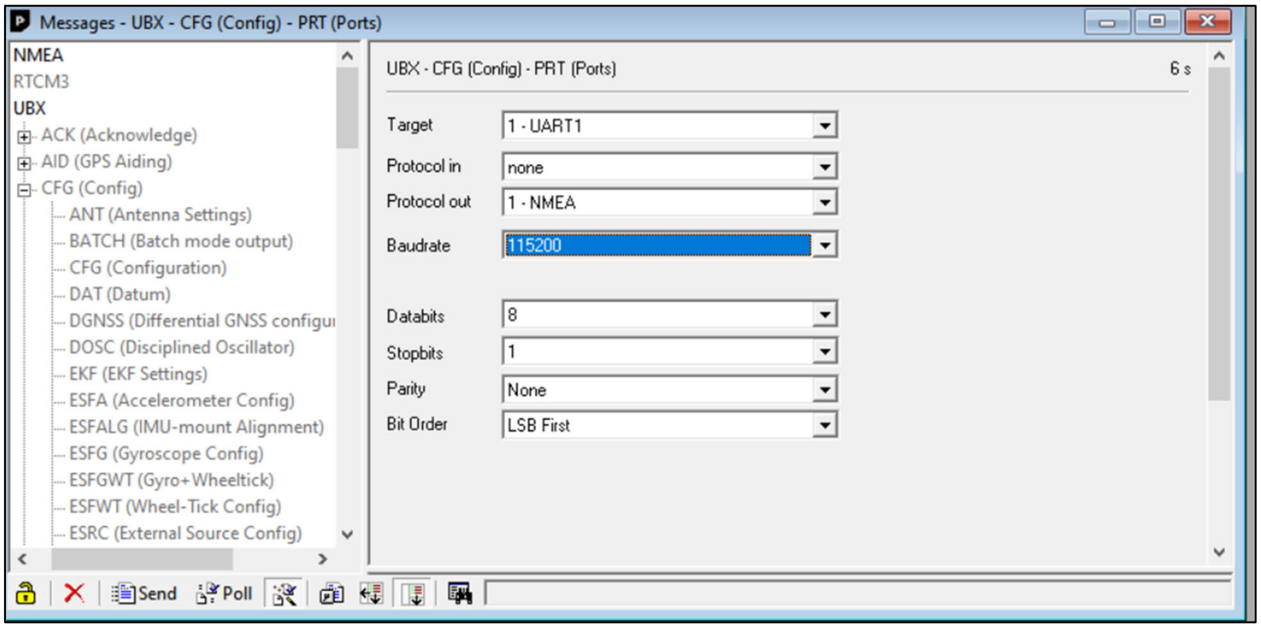
South GCP. Therefore, the known latitude, longitude, altitude, and accuracy (3 centimeters) of A1 South was entered into the TMODE 3 configuration fields.

Although the Rock Spring Orchard GCPs were surveyed by Penn State's Intelligent Vehicles and Systems Group, the ZED-F9P board can be used to establish a high-accuracy GCP. This procedure is described in Section 7.04.

For the configuration settings to remain on the board after reboot, one must save them to the battery-backed RAM and flash devices on the board. This can be done under View, Messages View, UBX, CFG, and again CFG. Saving a copy of the configuration settings to the PC is also helpful. This is done by selecting Tools, Receiver Configuration, the file destination, and then Transfer GNSS -> File.

The rover RTK module was configured to receive RTCM corrections through UART2 and output high-accuracy NMEA messages through UART1. The UART2 port (PRT) settings for the rover are identical to the base: RTCM3 in and out (noting that RTCM messages are only received by the board), baud rate 57600 bits per second.

While the UART1 settings are not relevant for the base module, this port is used on the rover module to send NMEA to the Pixhawk on the rover. For UART1, the output protocol was set to NMEA. The baud rate was set to 115200 bits per second, to accommodate the large number of messages and high message frequency. Figure 6.11 shows the UART1 settings for the rover module.


Messages - UBX - CFG (Config) - PRT (Ports)
NMEA
RTCM3
UBX
ACK (Acknowledge)
AID (GPS Aiding)
CFG (Config)
ANT (Antenna Settings)
BATCH (Batch mode output)
CFG (Configuration)
DAT (Datum)
DGNSS (Differential GNSS configu
DOSC (Disciplined Oscillator)
EKF (EKF Settings)
ESFA (Accelerometer Config)
ESFALG (IMU-mount Alignment)
ESFG (Gyroscope Config)
ESFGWT (Gyro+Wheeltick)
ESFWT (Wheel-Tick Config)
ESRC (External Source Config)
UBX - CFG (Config) - PRT (Ports)
6 s
Send
Poll

Target: 1 - UART1
Protocol in: none
Protocol out: 1 - NMEA
Baudrate: 115200
Databits: 8
Stopbits: 1
Parity: None
Bit Order: LSB First

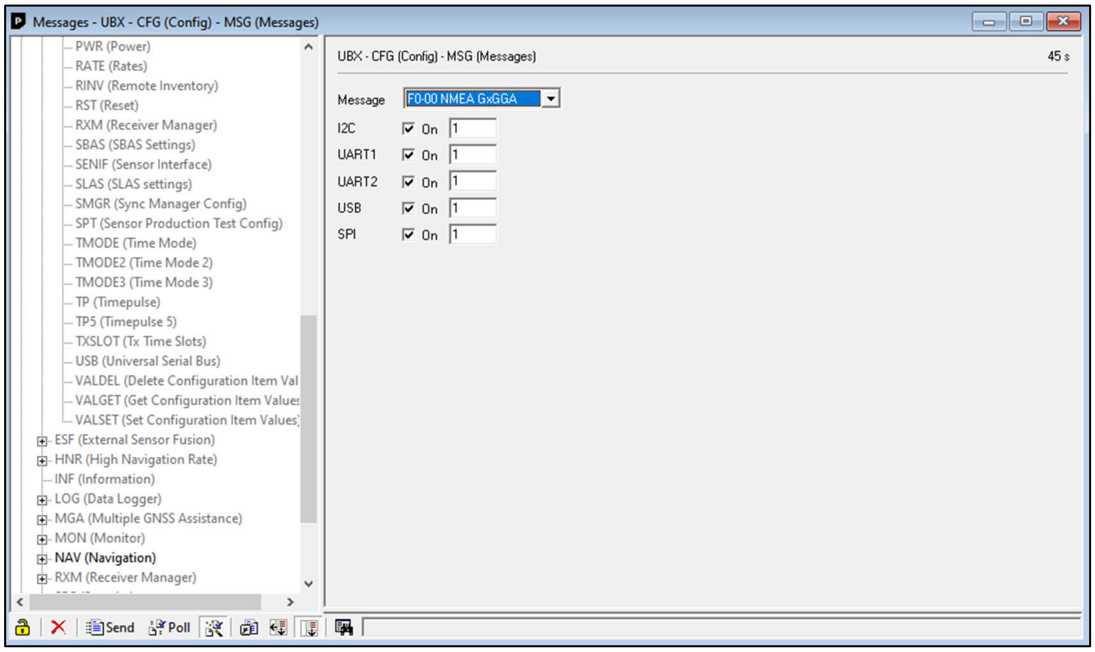
Figure 6.11 - RTK rover module UART1 configuration settings in u-center

The rover module was configured to send several NMEA message types. This configuration was done in u-center within the MSG settings. Table 6.7 summarizes the NMEA messages selected for UART1. The message contents of each message type were sourced from SiRF Technology's NMEA Reference Manual [33].

Table 6.7 - NMEA message types sent from UART1 of the rover RTK module

NMEA Message Type	*Message Contents*
GxGGA	GPS fixed data
GxGLL	Geographic position (latitude/longitude)
GxGSA	GNSS DOP and active satellites
GxGSV	GNSS satellites in view
GxRMC	Recommended minimum specific GNSS data
GxVTG	Course Over Ground and Ground Speed

Figure 6.12 shows an example of the rover module MSG configuration. The default port selection for NMEA output was left as all ports. The USB port is useful for troubleshooting and data logging.



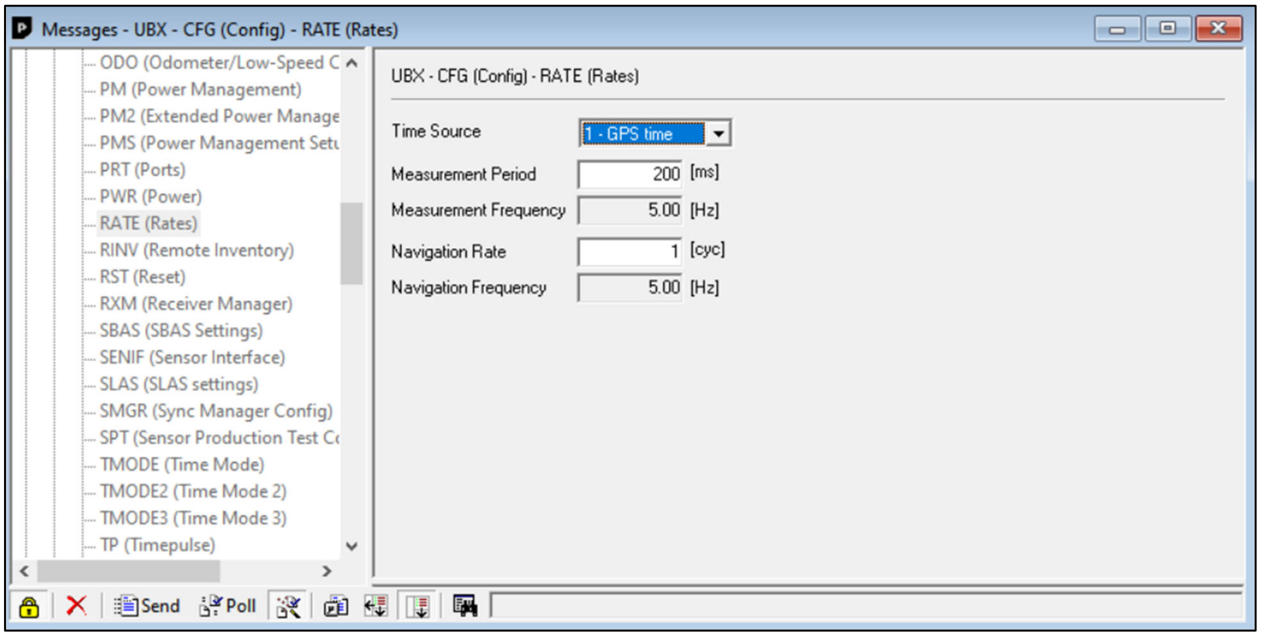
UBX - CFG (Config) - MSG (Messages) 45 s

Message F0-00 NMEA GxGGA

I2C	☑ On	1
UART1	☑ On	1
UART2	☑ On	1
USB	☑ On	1
SPI	☑ On	1

Figure 6.12 - U-center configuration of NMEA messages on the rover RTK module

The number fields to the right of the checkbox are not the periods of the NMEA messages, as it was with RTCM. Rather, the frequency of the NMEA messages is set in the RATE configuration menu, as shown in Figure 6.13. The measurement period for UTC (Coordinated Universal Time), GPS, GLO (GLONASS), BDS (Beidou), and GAL(Galileo) was set to 200 milliseconds. The navigation frequency must be increased to 5 Hz to make the NMEA output from the ZED-F9P compatible with the Pixhawk. The lowest position update rate allowed by Pixhawk is 5 Hz.



UBX - CFG (Config) - RATE (Rates)

Time Source 1 - GPS time
Measurement Period 200 [ms]
Measurement Frequency 5.00 [Hz]
Navigation Rate 1 [cyc]
Navigation Frequency 5.00 [Hz]

Figure 6.13 - U-center configuration of the rover RTK module navigation frequency

To fully harness the centimeter-level precision provided by the ZED-F9P, the latitude and longitude coordinates within the NMEA sentences must have a sufficient number of decimal places. The latitude and longitude within NMEA are formatted as degrees and minutes. By default, there are five decimal places (dd.mmmmm) thus the resolution is limited to 0.00001 minutes, or 1.855 centimeters at the equator. The ZED-F9P can be put into high-precision mode, increasing the number of decimal places to seven (dd.mmmmmmm). The resolution in high-precision mode is 0.01855 centimeters [24]. To enable high-precision mode one must navigate to View, Messages View, UBX, CFG, and then NMEA. The “High precision mode” and “Consider mode” flags must then be activated. The same procedure for saving the configuration settings of the base module must be followed for the rover module.

6.04 Configuring Pixhawk to Integrate the RTK System

After configuring the RTK base module and wiring it to the Pixhawk, ArduPilot needed reconfiguration to utilize the new source of location data. ArduPilot has the ability to incorporate two GPS devices, and therefore the ability to take in two streams of location data. By default, ArduPilot on the Pixhawk 4 wants the second GPS device to be wired to the Serial4 port. Since the RTK rover module was wired to Serial4, all ArduPilot parameters referring to the second GPS apply to the RTK module input.

The parameter changes required to integrate the RTK module are summarized in Table 6.8. The full list of parameters relevant to this chapter are found in Appendix A.2. The Serial4 port must first be assigned as a GPS input (by default, this is the case). The baud rate of Serial4 must match the 115200 bits per second output baud rate of the ZED-F9P rover module. ArduPilot must also know that the messaging protocol of the Serial4 port is NMEA. Knowing the quality of location data provided by the ZED-F9P is far better than that of the Pixhawk GPS module, the data stream from the GPS module can be ignored. This is done through the GPS auto switch parameter by opting to exclusively use the second GPS. Since the ZED-F9P is receiving satellite signals from four GNSS systems, these systems are bit-masked in ArduPilot. The ZED-F9P rover module was configured to update its position solution at 5 Hz, the minimum update rate allowable by Pixhawk. The update interval for the second GPS is set to 5 Hz. Given the superior 3-centimeter accuracy of the RTK system, the navigational tolerances for waypoints are constrained to 3 centimeters. The Extended Kalman Filter parameters in Table 6.8 set the GPS mode to 2D (since UGV01 will remain on the ground), increase the weight of the position provided by the RTK module, and set the lower accuracy limit of the RTK module to the value defined in the ZED-F9P datasheet [23]. Opting to ignore the input of the Pixhawk GPS module caused issues with the "GPS Configuration" arming check, so this check is disabled.

Table 6.8 - ArduPilot parameters required for ZED-F9P RTK module integration

Parameter	*Value [units]*	*Function*
SERIAL4_PROTOCOL	5	Assign Serial4 port as a GPS input
SERIAL4_BAUD	115200 [bps]	Set Serial4 baud rate to 115200 to match the RTK module's UART1 baud rate
GPS_TYPE2	5	Define that the messaging protocol of the RTK module is NMEA
GPS_AUTO_SWITCH	3	Exclusively use the RTK module for localization
GPS_GNSS_MODE2	77	Use GPS, GLO, BDS, and GAL satellite systems through the RTK module
GPS_RATE_MS2	200 [ms]	Set update interval for RTK module to 200 ms (5 Hz)
WP_OVERSHOOT	0.03 [m]	Constrain waypoint tolerance to the accuracy of the RTK module
WP_RADIUS	0.03 [m]	Constrain waypoint tolerance to the accuracy of the RTK module
EK2_GPS_TYPE	1	Define GPS control mode as 2D velocity and position
EK2_POSNE_M_NSE	0.1 [m]	Set GPS horizontal position noise to 10 cm to increase the weight of the RTK module measurements in position solutions
EK2_VELNE_M_NSE	0.05 [m/s]	Input lower limit of RTK module velocity accuracy
ARMING_CHECK	60926	Disable the "GPS Configuration" arming check

Chapter 7

Performance Evaluation of the RTK System

7.01 Overview

This chapter presents the methods used to test the precision and accuracy of the ZED-F9P base-rover pair at Rock Springs Orchard. In light of the improved precision achieved by the RTK system, a method of establishing a high-accuracy GCP an any geographic location is described. The performance of UGV01 with the RTK system at Rock Spring Orchard is presented and compared to the performance of the non-RTK configuration.

7.02 Assessing the Precision and Accuracy of the RTK System

The first tests of the RTK system were done with the RTK rover module removed from UGV01. The lightweight board with attached radio and GNSS antenna were easy to transport to and about Rock Springs Orchard when detached from the vehicle. Powering on the base module and connecting the USB port to a PC running u-center, it was verified that RTCM messages were being sent out. Powering on the rover module and connecting it to a PC, it was verified that the board had an RTK fix and high-precision NMEA messages were being sent out.

To evaluate the static precision and accuracy of the rover module, the RTK system was brought to Rock Springs Orchard. The RTK base station was set up at the A1 South GCP as shown in Figure 7.1. The GNSS antenna of the base station was placed at the center of the GCP paver via visual estimation. The rover RTK module, corrections radio, and GNSS antenna were brought to all six orchard GCPs. At each GCP the GNSS antenna was placed at the center of the paver via visual estimation and the RTK board was connected to a laptop running u-center. It should be noted that a Holybro radio was used on the rover module for these tests. However, there were no interruptions in the stream of corrections signals from the base, so the use of the lower-power radio did not affect accuracy or precision.

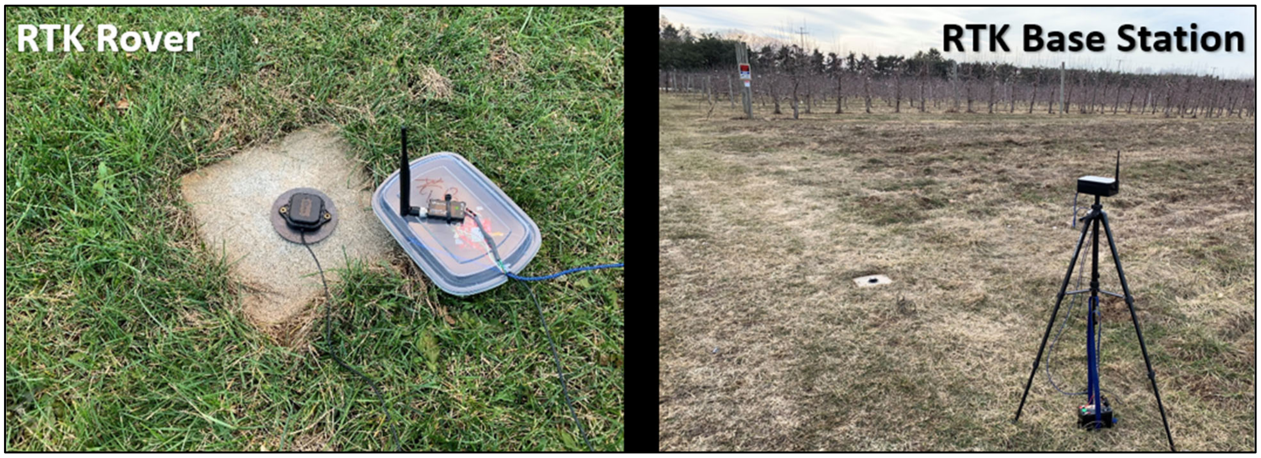
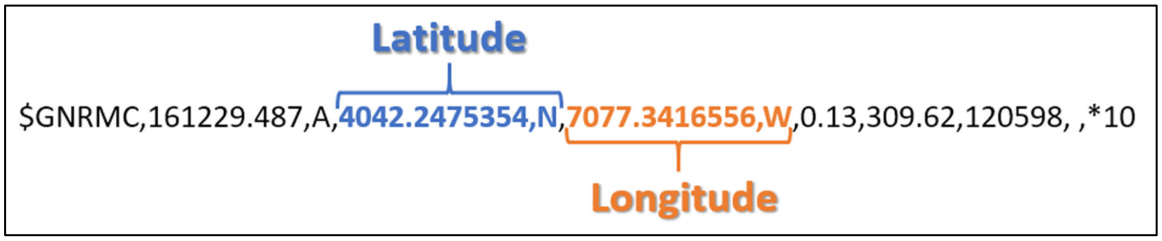


Figure 7.1 - RTK rover (left) and base station (right) equipment during accuracy testing

The recording function within u-center was used to log the messages sent out from the RTK module. The rover RTK module was configured to send location data to the Pixhawk in NMEA protocol. These NMEA messages were recorded to evaluate the accuracy and precision of the RTK module. Messages were recorded for about a minute at each GCP.

The raw NMEA sentences logged from the rover RTK module had to be processed to extract the latitude and longitude coordinates of each position solution. Among the NMEA sentence types output by the rover module at each update interval, position coordinates were extracted from the GNRMC sentences, as shown in Table 6.7. This was done by importing all logged NMEA sentences into a Microsoft Excel sheet, targeting GNRM sentences, and extracting all unique latitude and longitude coordinates. As shown in Figure 7.2, the latitude and longitude within a GNRMC sentence are in degrees and minutes (ddmm.mmmmmmm) and hemisphere is designated by cardinal direction. For ease of analysis, this format was converted to degrees latitude and longitude with hemisphere designation by sign (negative for south and west, positive for north and east). There are 60 minutes within one degree. For example:

ddmm.mmmmmmm, W = [dd+(mm.mmmmmmm/60)] * (- 1)



$GNRMC,161229.487,A,4042.2475354,N,7077.3416556,W,0.13,309.62,120598, ,*10

Figure 7.2 - Example of GNRMC NMEA sentence

After obtaining data sets of latitude and longitude coordinates at each GCP, statistical analysis was performed to obtain the standard deviation, mean latitude, and mean longitude. By comparing the mean coordinates to the known GCP coordinates, the average error was found. In an effort to produce more spatially meaningful results, the latitude and longitude statistics were converted into distance using the following equation:

$$d\ [ft] = 364813\sqrt{(lat2 - lat1)^2 + \left(cos\left(\frac{lat1 + lat2}{2}\right)(lon2 - lon1)\right)^2}$$

This conversion is based on the spherical Earth model provided by MathWorks (mean radius of 6371000 meters) [34]. This equation assumes 364,813 feet per degree of latitude and [364,813*cos(mean latitude)] feet per degree of longitude. Figure 7.3 shows the statistics for each GCP. The standard deviation of the rover RTK module was no more than 0.15 inches at any GCP. Therefore, the rover RTK module was found to be very precise. The average error at each GCP was no more than 4 inches. In Section 5.03, the allowable deviation from center was estimated to be 21 inches. Therefore, the accuracy of the RTK system is acceptable.

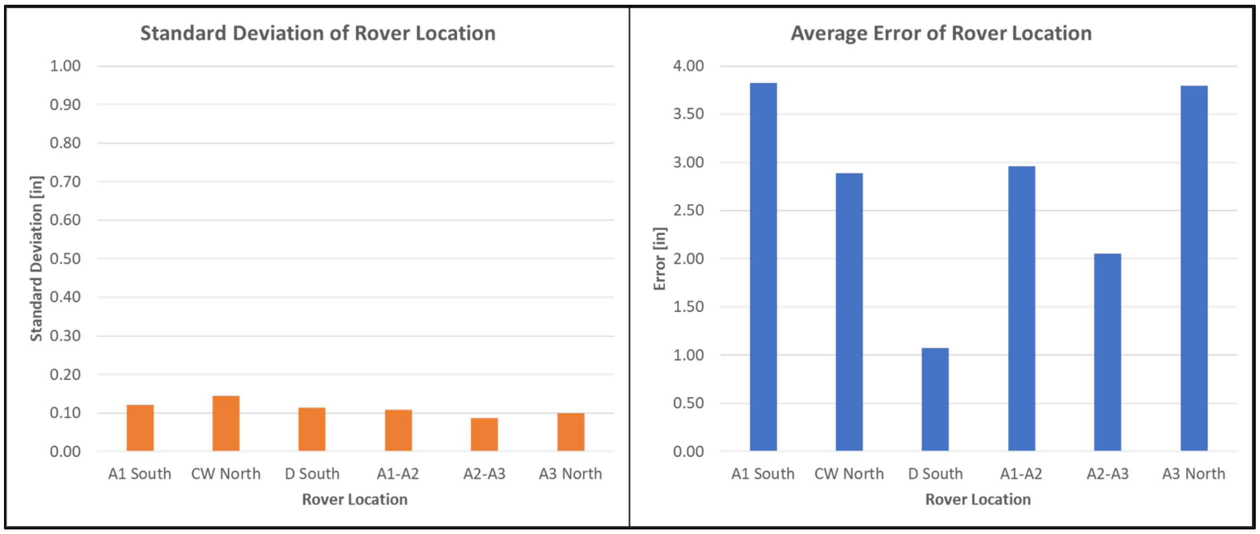


Figure 7.3 - Standard deviation and error of the rover RTK module at each GCP

One experimental factor to consider in the determination of accuracy is the visually estimated placement of the base and rover GNSS antennas. Misplacement of either would cause an uncontrolled experiment. This is seen in the error for the rover at A1 South. Since the base station antenna must be at the center of the GCP paver, the rover antenna could not be placed at the center, as shown in Figure 7.4. This misplacement is reflected in the 3.8-inch error for the rover RTK module at A1 South.

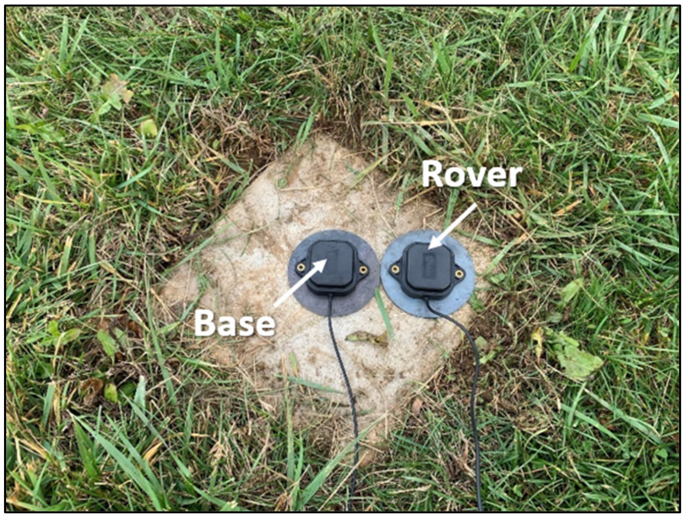


Figure 7.4 - Base and rover GNSS antennas sharing A1 South GCP during testing

7.03 Testing UGV01 with RTK-Integrated Pixhawk

After independently testing the RTK rover module, it was reconnected to the Pixhawk. UGV01 with the RTK-integrated Pixhawk was brought to Rock Spring Orchard to evaluate its autonomous performance.

To make the creation of Rock Spring Orchard missions more efficient, a Python script was written by Dr. H. J. Sommer. This script is named inline_pair_UGV01.py and is found in Appendix B.1. The script references a spreadsheet that contains the latitude and longitude of every support post. The support posts are at the ends of each orchard row. Since the orchard rows are straight the location, length, and orientation of each row is known. A user can input specific rows and the script will generate waypoints between these rows. The list of waypoints is saved as a tab-delimited *.waypoints file that is uploaded into Mission Planner. A user can also specify the spacing of these waypoints as well as the overshoot past the ends of the rows. Overshoot is necessary to navigate past the wires bracing the support posts. The reference spreadsheet also contains the locations of the test track secondary GCPs. Therefore, the script can be used to create test track missions.

The inline_pair_UGV01.py script was utilized to generate a test track mission for performance evaluation. UGV01 was to complete two loops around the rectangular test track, driving over the GCPs, overshooting them by 15 feet, and pivoting 90 degrees at the end of a pass. An intermediate waypoint was created at the halfway point of each long pass. This mission visualized in Mission Planner is shown in Figure 7.5.

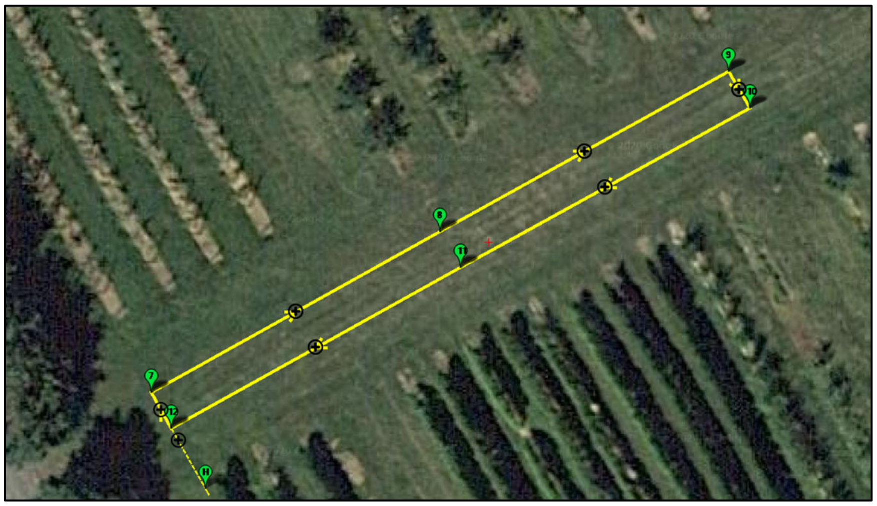


Figure 7.5 - Test track mission created for performance evaluations

To get a baseline performance of the non-RTK Pixhawk on UGV01, the ArduPilot parameters were reverted to the non-RTK configuration. The performance of UGV01 was evaluated by logging the location output from the ZED-F9P onboard the vehicle as it navigated the mission. This was done by connecting a long USB cable to the module and following UGV01 with a laptop running u-center. The recording function in u-center was used to log position data. Using the calibrated map feature on u-center, a precise and dimensionally consistent trace of UGV01 was plotted. For more information on the calibrated map feature, see section 7.04.

The position trace of UGV01 controlled by the non-RTK Pixhawk is shown in Figure 7.6. In this figure, the red circles mark the positions of the GCPs. The overlaid image of UGV01 shows the navigation direction of UGV01 and is not to scale. UGV01 missed some waypoints by several feet and the paths between waypoints were largely non-linear.

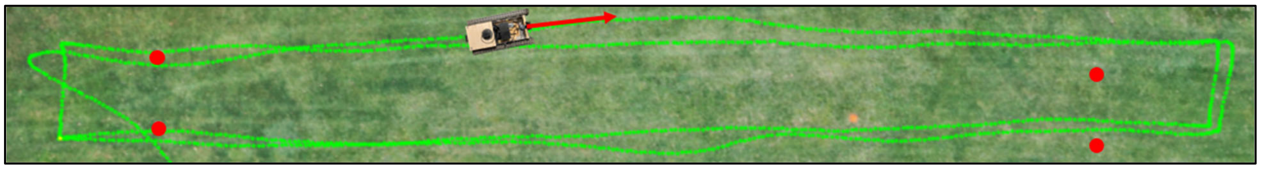


Figure 7.6 - Non-RTK Pixhawk navigating UGV01 through test-track mission

The position trace of UGV01 with the RTK Pixhawk is shown in Figure 7.7. The RTK system greatly improved the performance of the Pixhawk. Paths between waypoints were straight with little deviation. The center of UGV01 came within inches of the center of each GCP.

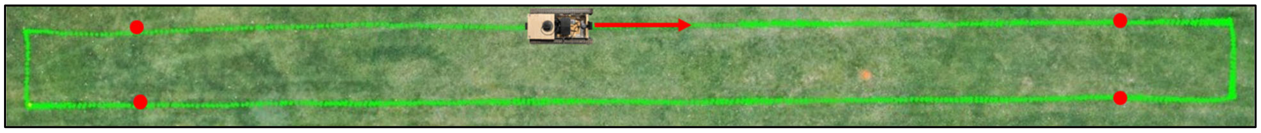


Figure 7.7 - RTK Pixhawk navigating UGV01 through test-track mission

The largest waypoint deviation was at A1-A2 NW, where the center of UGV01 passed over the edge of the paver. The locations of the pavers are known by photogrammetry and are therefore accurate to 14 centimeters, or 5.5 inches. The 6-inch deviation from the center of the paver could be the result of the photogrammetry inaccuracy. Figure 7.8 and Figure 7.9 show closeup views of UGV01's position at each end of the test track. UGV01 had an occasional tendency to veer right when moving after a complete stop or pivot turn. This behavior is likely caused by the non-neutral timing of the motors causing the left motor to spin faster than the right at low speeds. Although the PWM parameters were tuned to counteract this effect, it cannot be entirely eliminated. Regardless, the Pixhawk is able to correct for the initial heading error shortly after getting up to cruise speed.

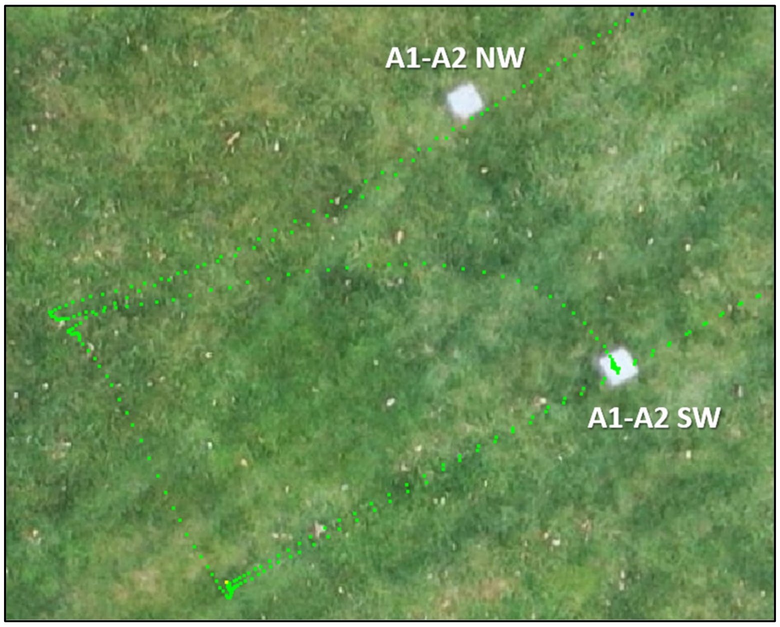


Figure 7.8 - Position trace of UGV01 with RTK Pixhawk: west end of test track

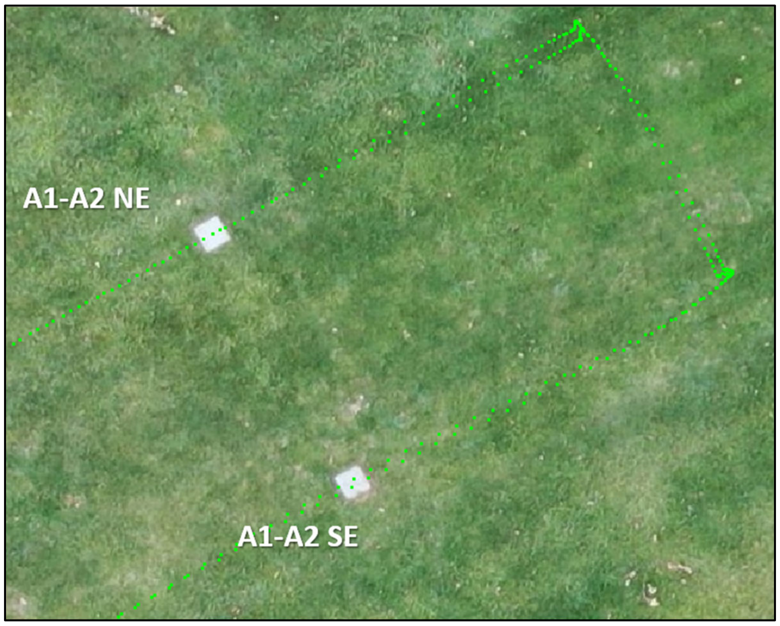


Figure 7.9 - Position trace of UGV01 with RTK Pixhawk: east end of test track

After a successful performance demonstration of the RTK Pixhawk on the test track, a mission was created to navigate UGV01 through several orchard rows. The inline_pair_UGV01.py script was used to generate waypoints between the first five westmost rows of orchard block A1. The mission avoids the large aisle between the third and fourth rows, thereby creating three passes through the trees. Of all the rows at Rock Springs Orchard, block A1 has the narrowest spacing between rows (about 8 feet). Narrower row spacing leaves less room for UGV01 to deviate from the center. Testing within the most difficult circumstances makes the results applicable to all other rows of the orchard. During this test there were many branches laying on the ground. When UGV01 climbed over the branches, some deviation from the centerline occurred. This is best seen at the southern end of the westmost row in Figure 7.10. The black arrows in this figure show the centerline between the trees as well as the direction of travel. Green lines that are seen beside the black centerline are deviations from the center.

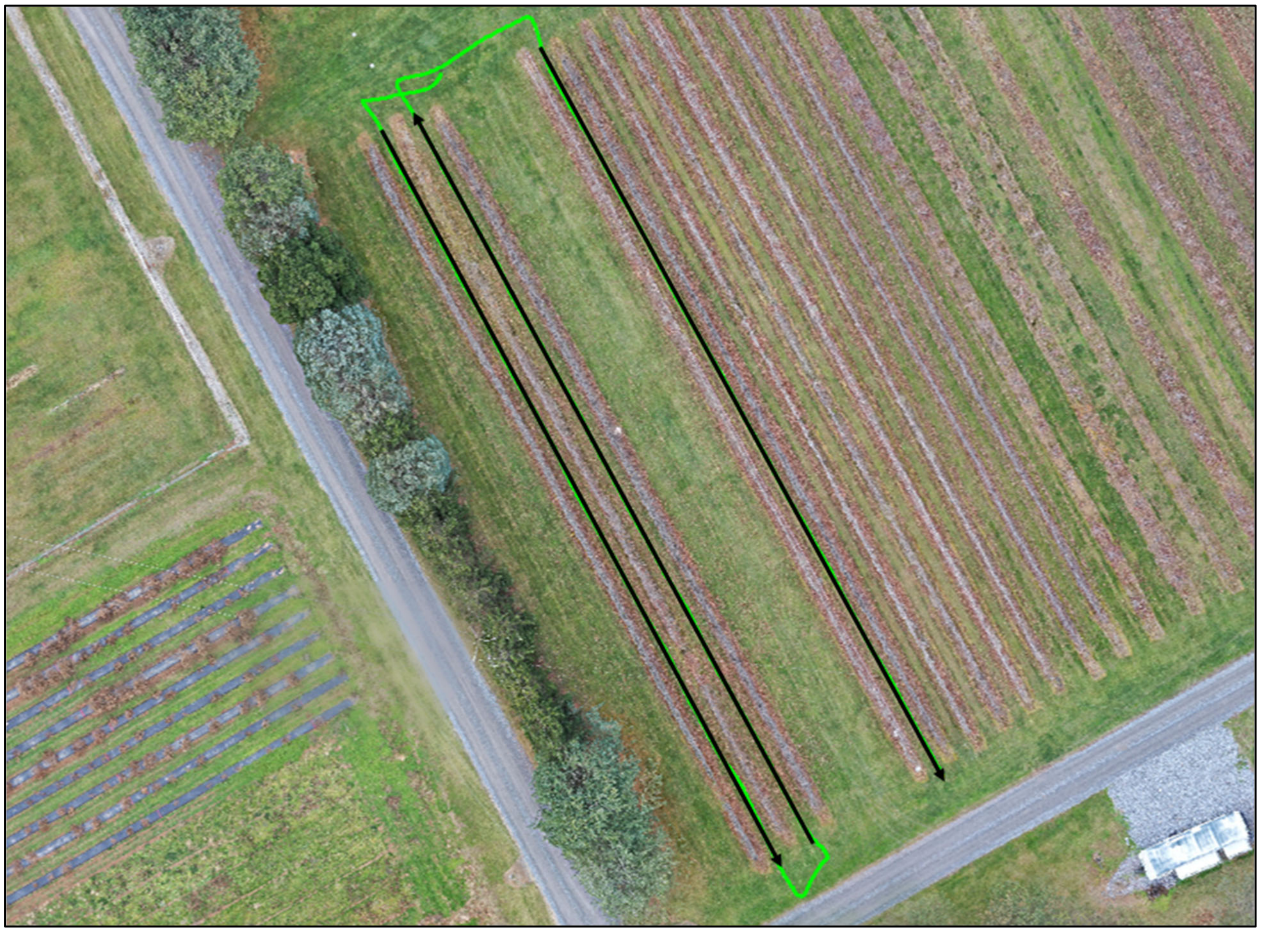


Figure 7.10 - Position trace of UGV01 through orchard block A1

When navigating through orchard rows, the GNSS antenna on UGV01 has a more obstructed view of the sky, as shown in Figure 7.11. Therefore, it is more challenging to “see” satellites and produce a high-precision location. To ensure the positional accuracy of the ZED-F9P did not falter when flanked by trees, the statistics recorded by u-center were reviewed. During the orchard mission, an RTK fix (highest precision mode) was achieved for 92% of the mission, an average of 32 satellites were in view, and the average precision was 1.6 centimeters. Based on these promising results, it was concluded that the RTK module did not have an issue producing high-precision solutions while flanked by trees. Furthermore, the system had satisfactory performance and was ready to be ported onto the Cub Cadet RZT-S Zero.

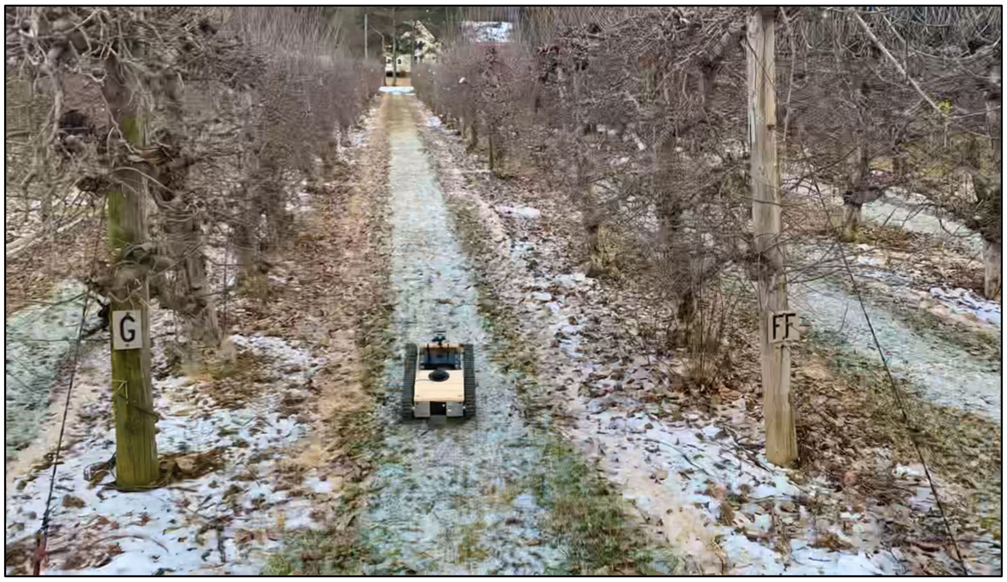


Figure 7.11 - UGV01 navigating a narrow orchard row in block A1

7.04 Using the ZED-F9P to Establish a High-Accuracy GCP

The high precision of the ZED-F9P RTK board with an L1/L2 GNSS antenna makes it a viable tool for establishing a GCP. Logging the data received from GNSS satellites over a long time period allows for calculation of a millimeter-level static location. Nathan Seidle has published helpful documentation on using the ZED-F9P to establish a GCP [27]. If Rock Springs Orchard did not have pre-surveyed GCPs, the ZED-F9P could have been used to find their precise locations. To demonstrate this ability, the ZED-F9P was used to establish a GCP at an off-site residential location.

The GNSS antenna was first mounted at the location of the GCP (on top of the roof of the residence). The grounding plate and antenna were adhered to the roof and their location was marked with a white wax pencil as shown in Figure 7.12.

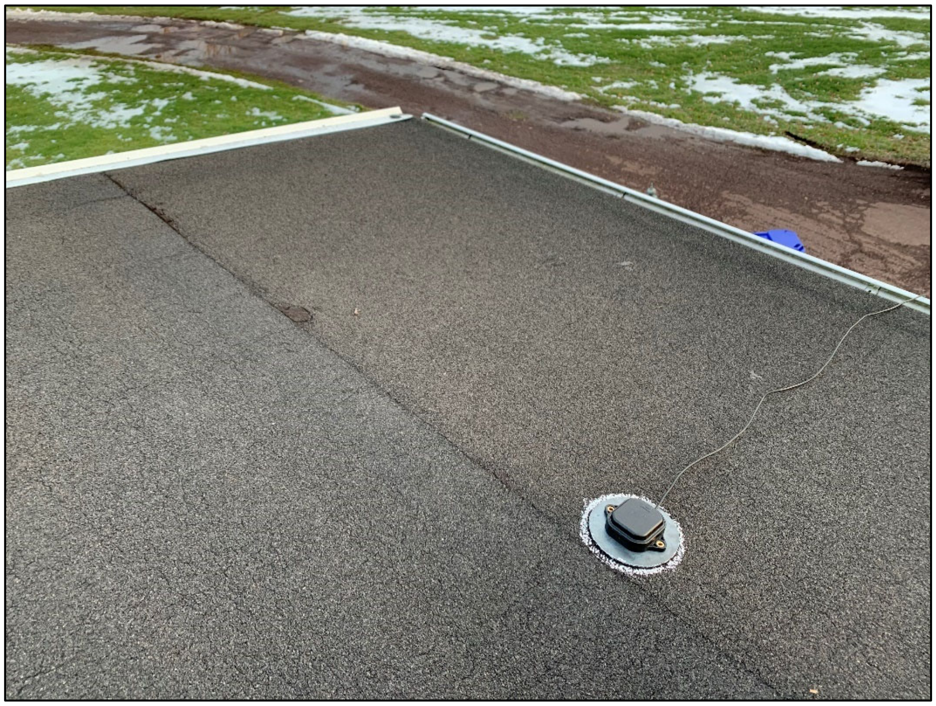
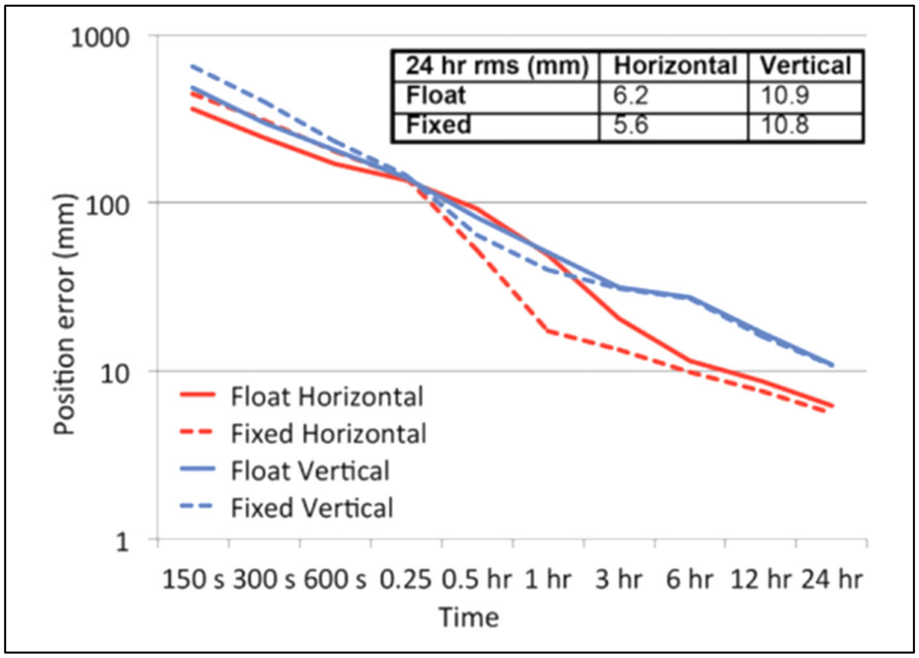


Figure 7.12 - GNSS antenna mounted on roof to determine GCP location

The ZED-F9P board was then configured to collect raw data from the GNSS antenna. In u-center, this is done by navigating to View, Messages View, UBX, CFG, and then MSG. The 02-15 RXM-RAWX message type was enabled for USB. According to Robot Operating System (ROS) documentation, the u-blox RXM-RAWX message type contains pseudorange, Doppler, carrier phase, phase lock and signal quality information for satellites [35]. Using the record feature in u-center, the raw GNSS data was collected for 15.75 hours. Noting the plot in Figure 7.13 created by Suelynn Choy [36], GNSS precise point positioning error falls logarithmically as data collection time increases. After 12 hours, the error in the position of the antenna is less than 10 millimeters.



24 hr rms (mm)	Horizontal	Vertical
Float	6.2	10.9
Fixed	5.6	10.8

Figure 7.13 - Decrease of position error with logarithmic increase of GNSS data collection time [36]

The raw data file was then converted into a *.obs file using the latest version of RTKCONV [37] and then compressed. The raw data was submitted to the Canadian Government's Precise Point Positioning (PPP) service [38]. First an account was created with the service. Then ITRF was selected as the processing mode. Finally, the zipped *.obs file was selected and submitted.

The Canadian PPP service processed the raw data and provided a report with the estimated coordinates of the GNSS antenna. The latitude and longitude were provided with an accuracy of +/- 0.003 meters. This is an order of magnitude more accurate than the locations of the orchard GCPs. Therefore, it was concluded that the ZED-F9P is capable of establishing a high-accuracy GCP.

For applications of this work where the ZED-F9P RTK system must be used to map several GCPs and orchard rows, it is not feasible to log data for 15 hours at every location. After establishing a base GCP with 3-millimeter accuracy, the rover RTK module can be used to determine other GCP locations. The accuracy of the rover module is limited by its reported precision, which was about 2 cm. Therefore, additional GCPs and landmarks can be determined to an accuracy of 2 cm.

At the off-site residential location, drone imagery and photogrammetry were not available. Instead, publicly available satellite imagery was used to create a calibrated map in u-center. The satellite image was sourced from Google Earth Pro [39]. A high-resolution image was exported from this program by selecting File, Save, and Save Image. Within the Save Image view, all overlays were removed and resolution was set to maximum, as shown in Figure 7.14.

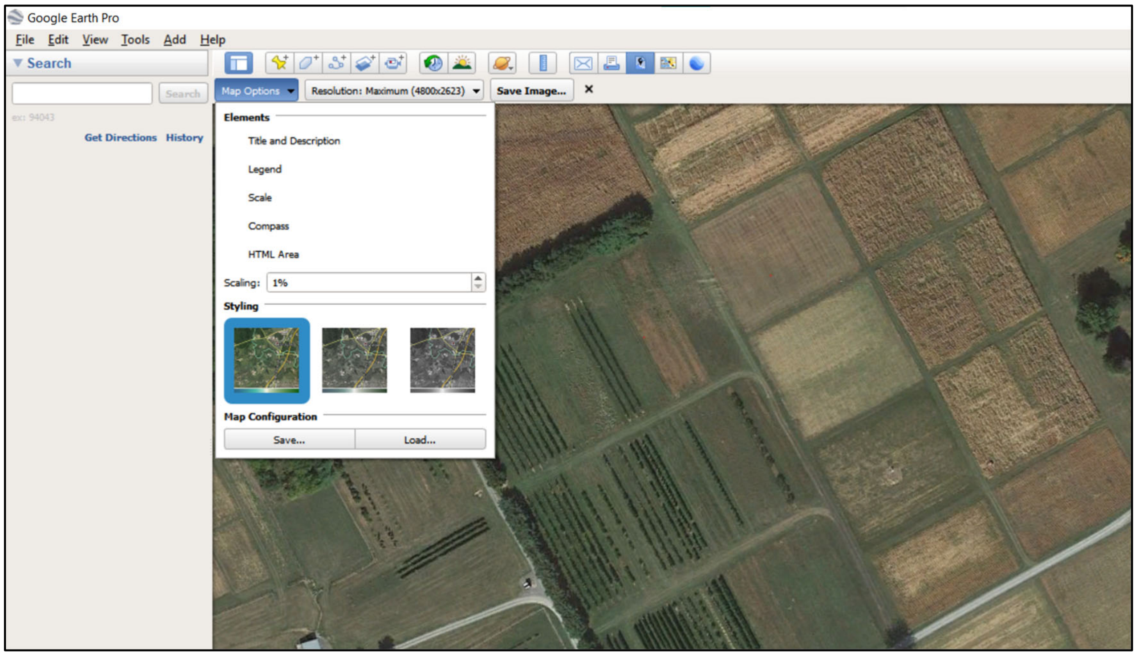

Resolution: Maximum (4800x2623)
Scaling: 1%

Figure 7.14 - Exporting a high-resolution satellite image from Google Earth Pro

To calibrate the high-resolution satellite image, the locations of visible landmarks must be known. Thus, GCPs must be established on landmarks that have been captured by the satellite imagery available on Google Earth Pro. U-center offers a tool to create a calibrated map from any image. The tool maps pixel coordinates to three known latitude and longitude coordinates across the image. The tool is accessed by navigating to View, Map View, and then the folder icon to open an image or calibrated map. If an image without a calibration file in the same directory is selected, u-center will prompt the user to calibrate the map. This entails selecting three points on the map and inputting the geographic coordinates for each. After completing the calibration, the map can be used to find the geographic coordinates of any point on the map with an estimated accuracy of 10 inches.

Chapter 8

Integrating ROS with the Pixhawk

8.01 Overview

This chapter describes the work done to integrate Robot Operating System (ROS) into the Pixhawk ground control system. The motivation of the work is based on designs for future autonomy systems at Rock Spring Orchard. With ROS, an automated mission-updating feature and simple object avoidance routine were developed.

8.02 Motivation for ROS

Integrating the RTK system with the Pixhawk on UGV01 showed the Pixhawk can successfully control a ground vehicle through Rock Spring Orchard with only satellite-based localization. While one autonomous vehicle brings some utility to an orchard, a team of autonomous vehicles could be used to achieve more robust tasks. An example of a more robust task would be mitigating frost damage to trees. To accomplish this, one high-altitude unmanned aerial vehicle (UAV) would use vision to map temperature across the entire orchard. A low-altitude UAV would use vision to focus on one specific block of the orchard. A UGV with a heater would be ready to mitigate frost in cold spots identified by the UAVs. With this topology, the autonomous team must be able to communicate with each other and share senor data.

ROS is a Linux-based middleware with the message-passing capabilities needed to create a system of autonomous vehicles. With ROS running on a ground control station for each autonomous vehicle, a network of communication between the vehicles can be created, as shown in Figure 8.1.

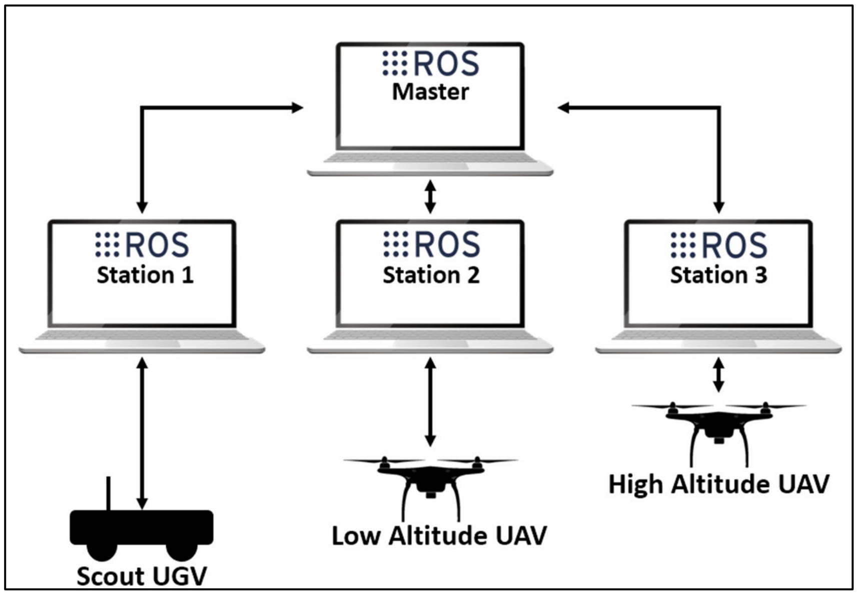


Figure 8.1 - ROS communication network for a team of autonomous orchard vehicles Laptop image from T. Ishikawa [40]

UGV01 with its RTK-integrated Pixhawk demonstrates the ability of Pixhawk to guide a ground vehicle through the orchard with satellite-based localization. Without ROS, one must use Mission Planner on a Windows PC to communicate with the Pixhawk. This prevents the default ground control system from being compatible with the multi-vehicle communication network. The ground control station for UGV01 must be able to communicate with the autopilot system through ROS.

8.03 Using ROS to Communicate with the Pixhawk

The ground control station must be a Linux PC to run ROS. Therefore, a laptop running Ubuntu 20.04.2.0 (Linux-based distribution) was acquired for the ground station. Christopher

Hirsh—a Penn State Mechanical Engineering Department System Administrator—assisted in building the Ubuntu laptop and installing the latest distribution of ROS. At the time of installation, this distribution was Noetic Ninjemys. The wiki site for ROS provides excellent documentation for tutorials and packages [41], including instructions for the installation of ROS on an Ubuntu PC.

It is helpful to understand the concepts of the ROS computation structure before describing the specific Pixhawk-ROS application. ROS Nodes are processes that perform computation. An example of a ROS node is a process that produces a location solution from raw GNSS data. ROS messages are the data structures that nodes use to communicate with each other. Nodes send and receive messages by publishing and subscribing to ROS topics. Multiple nodes can publish to a single ROS topic; similarly, multiple nodes can subscribe to a single topic. With the publish/subscribe messaging system, the nodes are unaware of other nodes' existence, thereby separating information generation from information reception. ROS services are an alternate communication method that follows a request/reply format. ROS services are appropriate for distributed-computing communication (multiple ROS computers). ROS master is required for nodes, messages, and services to work together. Essentially, ROS master allows the components of the computation network to find each other and work together. ROS bags are used for logging message data [42].

Pixhawk sends and receives messages from a ground control station with Micro Air Vehicle (MAV) Communication Protocol, commonly called MAVLink. MAVLink is designed for communication to drones from the ground station as well as communication among components onboard a drone. Similar to ROS topics, MAVLink follows a publish/subscribe messaging pattern for all messages, with the exception mission plan and parameter sub-protocols

[29]. Mission plans and parameters are downloaded/uploaded using point-to-point communication. This sub-protocol supports re-request and retransmission of messages not received. This is an important design feature for message transmission via radio telemetry because connection deficiencies can cause message losses [43].

MAVLink must be used to communicate with the Pixhawk. Accordingly, Mission Planner uses MAVLink protocol to send commands, parameters, mission plans, etc. to the Pixhawk. For ROS to assume all functionality of Mission Planner while adding flexibility and robustness to the ground control system, it must be able to send and receive messages using MAVLink protocol. MAVLink message transmission was achieved in ROS by installing the MAVROS package, authored by Vladimir Ermakov [44]. MAVROS has three nodes: main communication node; ground control station (GCS) bridge node; and event launcher node. The GCS bridge node uses User Datagram Protocol (UDP) to pass all messages received by MAVROS to Mission Planner. Telemetry radios were used to establish the UDP bridge between MAVROS on Linux and Mission Planner on Windows. The topology of this system is shown in Figure 8.2.

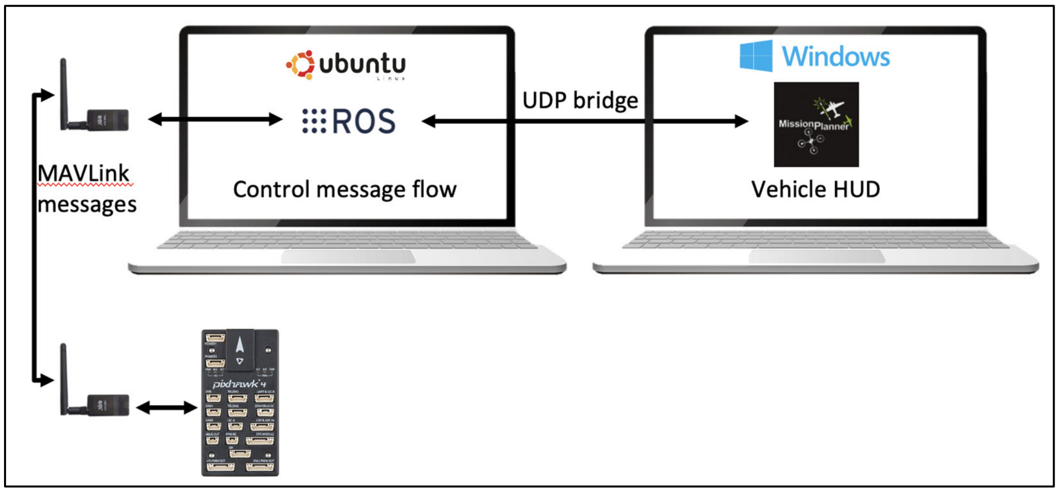


Figure 8.2 – Topology of Pixhawk, ROS, and Mission Planner

8.04 Assigning Missions to UGV01 with MAVROS

The team of interconnected autonomous orchard vehicles must be able to provide mission updates to each other through ROS. For example, if a UAV needs the UGV to address an issue in a specific orchard block, the mission of the UGV will be updated to navigate through that block. To make UGV01 compatible with this team configuration, a mission-updating Python script was developed for ROS. The name of this script is mission_update.py and can be found in Appendix B.2. It is run on the Linux PC in the command window while ROS master and MAVROS are running and the Pixhawk is successfully communicating with MAVROS.

The script was designed to be a mission intervention routine; one that could be run while UGV01 was actively completing a mission. Consequently, the script first stops UGV01 by putting the Pixhawk in hold mode. Hold mode sends neutral PWM outputs (about 1500 microseconds) to the left and right motors, stopping the vehicle. Mode changes are done with MAVROS by calling the “set_mode” service of the “sys_status” plugin.

After stopping UGV01, the current waypoint list is cleared from the Pixhawk. The “clear” service of the “waypoint” plugin is called to clear the list.

The inputs of mission_update.py are identical to those of inline_pair_UGV01.py: orchard rows, waypoint spacing, and overshoot distance. In fact, the waypoint generation code written by Dr. Sommer was integrated into mission_update.py so that the script could generate the new mission that is downloaded to UGV01. With this design, if another autonomous vehicle is connected to UGV01 with ROS, the vehicle can specify a row of the orchard and the latitude/longitude waypoints will automatically be generated. The inline_pair_UGV01.py script is designed to generate a tab-delimited *.waypoints file for Mission Planner to read. Since Mission Planner is no longer required to communicate with the Pixhawk, this *.waypoints file is no longer necessary. Instead, the mission_update.py script creates a waypoint list array that stores each waypoint. The information for each waypoint is stored in an object. The inline_pair_UGV01.py script iterates to write each waypoint's information to the *.waypoints file. The mission_update.py script iterates to append the waypoint list with each waypoint's information.

After the script finishes appending the waypoint list array with each waypoint object, the mission is downloaded to the Pixhawk. The "waypoint" plugin "push" service is called to download the waypoint list to the Pixhawk.

Before initiating the new mission, the Pixhawk is told to restart its waypoint sequence. That is, the current waypoint must be reset to the first waypoint. The current waypoint is set using the "set_current" service of the "waypoint" plugin.

The first waypoint of the waypoint list (of index zero) indicates the "home" location of UGV01 and is not interpreted as a waypoint to achieve. When creating the waypoint list, the mission_update.py script stores empty values for index zero. After downloading the new waypoint list, the home location is set to the current location of UGV01. This is done with the "set_home" service within the "command" plugin.

The last action performed by mission_update.py is the reactivation of auto mode. Auto mode initiates the new mission downloaded to UGV01 and is set by calling the "set_mode" service of the "sys_status" plugin.

8.05 Simple Object Avoidance with ROS and Pixhawk

ROS provides the framework for interpreting data from multiple sensor inputs to improve localization. To apply this framework to the Pixhawk, a simple ultrasonic sensor was used to feed range sensor distances to ROS. The added hardware consisted of an Arduino Uno with an HC-SR04 ultrasonic sensor.

The Arduino had to be configured to interpret the signal from the sensor as well as publish range values to a topic in ROS. The full code for the Arduino Uno is found in Appendix C.1. The distance is calculated by dividing the time it takes to echo the ultrasonic pulse by the speed of sound. The Arduino sketch creates a topic named “ultrasound” to which it publishes a message of type “range_msg.” This message type is established within the standard library of sensor messages in ROS. The node that handles the ultrasonic topic is “serial_node.” This node must be initiated in ROS master for the range messages to be available.

A Python script was developed to subscribe to ultrasound topic and send commands to the Pixhawk based on distance. This script was named backup.py and is found in Appendix B.3. The script is designed to imitate an object-avoidance routine followed by ROS for a front-facing ultrasonic sensor onboard UGV01. The script establishes a node for publishing and subscribing. It creates a subscriber for the range messages on the ultrasound topic. It also creates a publisher for the “OverrideRCIn” messages on the MAVROS “override” topic. OverrideRCIn messages override input signals from the DX8 R/C controller connected to the Pixhawk. When the range messages fall below 30 centimeters, the R/C signals are overridden to cause UGV01 to backup. Channel 3 in ArduPilot controls throttle, thus a value of 1300 microseconds (200 less than neutral output) is used to move the left and right tracks in reverse.

Chapter 9

Cub Cadet R/C Control

9.01 Overview

This chapter describes the electrical system for manual control of the Cub Cadet RZT-S Zero and a custom microprocessor circuit to allow the RZT-S to be remotely operated by standard radio control (RC) signals. This approach allows the RZT-S to be integrated with Mission Planner and controlled from a remote base station. The authorship of this chapter, and the work herein, is accredited to Dr. H.J. Sommer.

9.02 Manual Control System

The RZT-S has four main electrical subsystems as shown in Figure 9.1 and Figure 9.2 from the shop manual [45] - vehicle control module (VCM) and manual input sensors, control panel, drive and deck motors, and batteries.

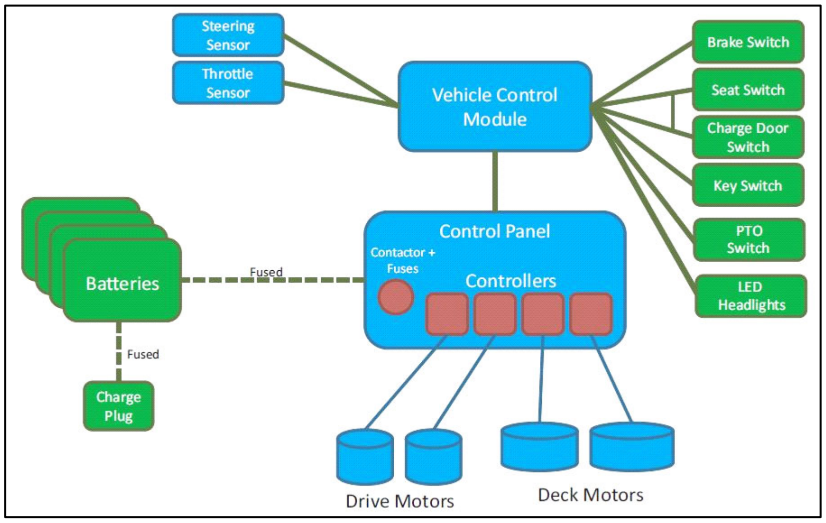


Figure 9.1 - RZT-S Zero electrical block diagram

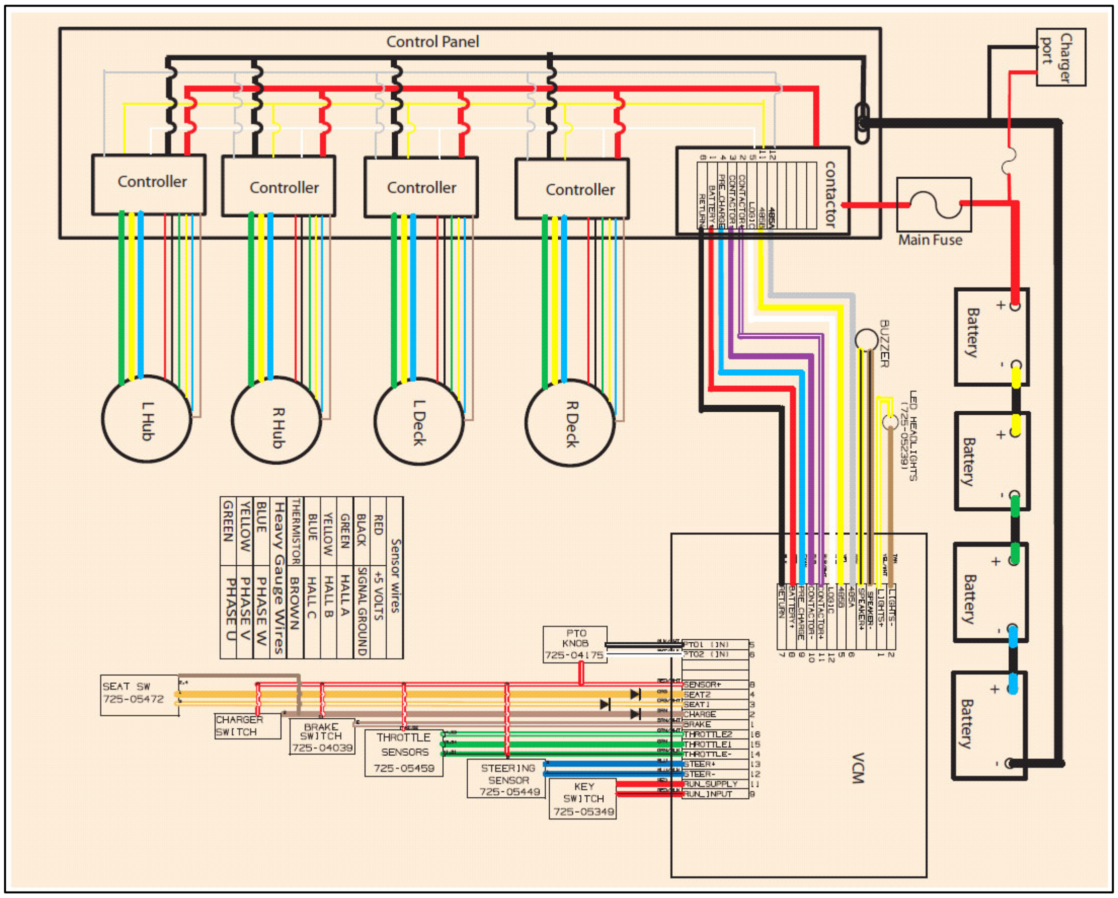


Figure 9.2 - RZT-S Zero electrical schematic

The VCM is a central computer that controls everything on the mower and acts as a user interface. It is housed in a plastic module on the steering column as shown in Figure 9.3. The VCM is connected to the control panel by an 8-wire cable into a 12-pin connector (Molex 0334721206) on the contactor as shown in Figure 9.2. The VCM will be replaced by a custom microprocessor circuit.

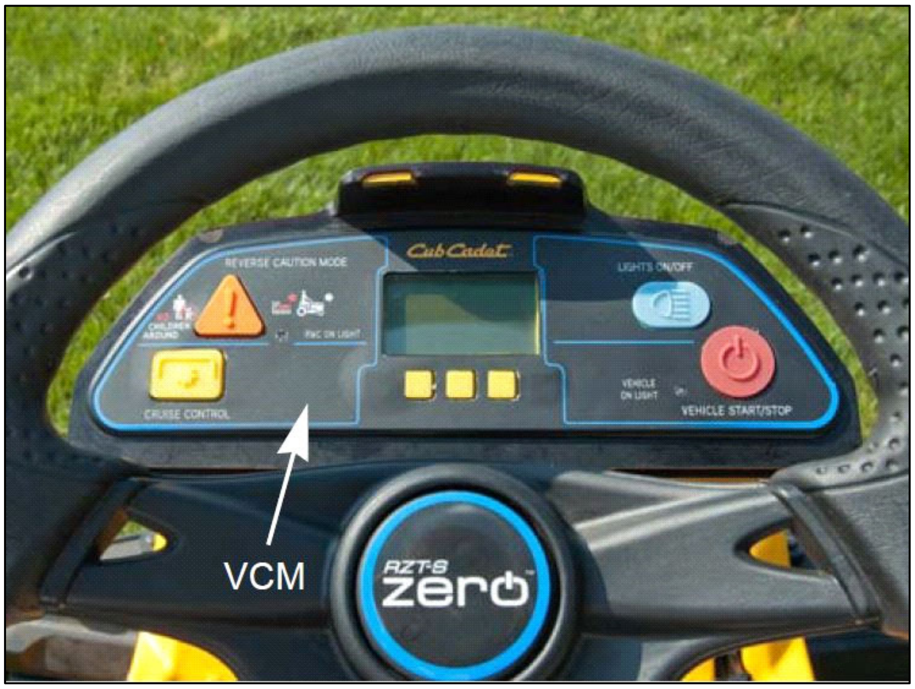


Figure 9.3 - RZT-S Zero Vehicle Control Module (VCM)

The control panel is mounted over the left rear wheel as shown in Figure 9.4 and consists of a contactor assembly and four BAC1000 motor controllers from Accelerated Systems Inc. (ASI). The contactor is mounted on the bottom of the panel and the BAC1000s are mounted on the top. The contactor is a large relay similar to a starter solenoid that allows a low current 48 VDC 0.3 A coil signal from the VCM to energize the relay and provide high current capacity between the battery pack and the four motor controllers. There is a 15 A fuse inside the contactor to protect the coil. Logic signals from the VCM are passed directly through the control panel to all four motor controllers using a 2-wire daisy chain RS-485 serial computer bus and a LOGIC enable line.

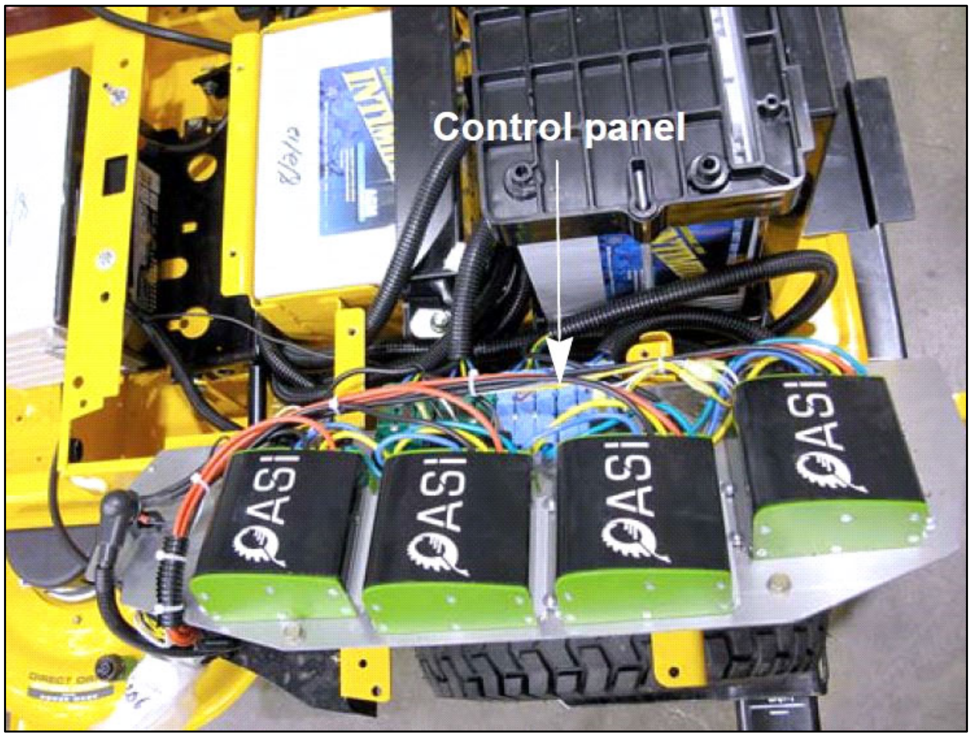


Figure 9.4 - RZT-S Zero control panel

The RZT-S has four brushless electric motors to drive the rear wheels and mower blades - Lhub for left rear wheel, Rhub for right rear wheel, Ldeck for left mower blade and Rdeck for right mower blade. Hub motors are rated 48 VDC at 11A with 200 W providing 1500 rpm. Deck motors are rated 48 VDC at 30 A with 1200 W providing 3000 rpm.

There are four 12 VDC sealed deep cycle lead acid batteries wired in series to provide 48 VDC power in an isolated fully floating system. This means that the electrical system is not grounded to the chassis and is insulated from non-electrical components of the mower. The battery subsystem includes a 150 A main fuse in-line between the battery positive cable and contactor. There is also a 20 A charger fuse.

It should be noted that the Ldeck motor on the old RZT-S has a Hall sensor fault and is not functional.

9.03 BAC1000 Motor Controllers

Each brushless motor is controlled by a BAC1000 controller [46]. Connections between each motor and its BAC1000 follow standard labelling commonly used for brushless motors. Power connections are Phases U/V/W. Signal connections include Hall A/B/C, 5 VDC power, signal ground and a thermistor to measure motor temperature.

The VCM sends digital message packets simultaneously to all four BAC1000s using a bidirectional two wire RS-485 serial bus at 115200 baud. Message packets conform to standard Modbus protocol [47] and must include a cyclic redundancy check checksum. The two RS-485 bus wires (0 to 5 VDC) and a 48 VDC LOGIC enable wire are daisy chained to all BAC1000s within the control panel.

A BAC1000 is controlled by writing/reading values to/from internal registers that manage motor drive circuits and provide controller status. Each register has a unique register address between 0 to 511. All registers hold 16-bit values. Writing or reading a register is called a command. An object dictionary of BAC1000 commands and their corresponding register addresses was provided by ASI [48] in XML format. Commands can write or read multiple contiguous registers at the same time.

Message packets from the VCM must contain write or read commands for a specific register in a specific motor controller. Each message starts with a one-byte hexadecimal (hex) identifier (ID) for a specific motor, a one-byte hex value 0x10 for write or 0x03 for read, and two bytes for the register address. Hex IDs for motors are Lhub 0x10, Rhub 0x13, Ldeck 0x16 and Rdeck 0x17. These IDs are defined by hardwired jumpers inside 16 pin connectors on each BAC1000.

Common write commands and their register addresses include write command timeout threshold 32, write remote speed 490, write remote maximum motoring current 491, write remote maximum braking current 492 and write remote state 493.

Common read commands and their register addresses include read command timeout threshold 32, read faults 258, read motor speed 264, read battery voltage 265 and read remote state 493.

The exact format for commands is defined by the Modbus protocol [47]. The VCM is master and all four BAC1000 are slaves on the RS-485 bus. A read command uses Modbus Section 3.3 "Request (Master to Slave)" format. Each valid request will receive a Modbus Section 3.4 "Valid Response (Slave to Master)" reply. An invalid request will receive a Modbus Section 3.5 "Error Response (Slave to Master)" reply. A write command uses Modbus Section 3.7 "Request (Master to Slave)" format. Similarly, valid requests will receive a Modbus Section 3.8 "Valid Response (Slave to Master)" and invalid requests will receive a Modbus Section 3.9 "Error Response (Slave to Master)".

Actual commands from the VCM to BAC100s for manual operation were reverse engineered using an RS-485 to USB module coupled in parallel with the RS-485 bus at one of the BAC1000 connectors. The RZT-S was lifted onto blocks with the rear wheels free and manual driving was emulated while recording the RS-485 bus hex communication stream. The hex stream was then decoded using the ASI object dictionary. Interestingly, the VCM only used two commands for all motors - one to set four contiguous registers 490-493 and one to read seventeen contiguous registers 256-272 as shown in Table 9.1 and Table 9.2.

Table 9.1 - VCM write registers for manual control

Register Address	*Command*	*Valid Range for Signed 16b Register*
490	remote speed	0xF000 = -4096 = -100% speed 0x0000 = 0% speed 0x1000 = +4096 = +100% speed
491	remote maximum motoring current	0x0000 = 0% current 0x1000 = +4096 = +100% current
492	remote maximum braking current	0x0000 = 0% current 0x1000 = +4096 = +100% current
493	remote state	0x0001 = 1 = idle 0x0002 = 2 = run

Table 9.2 - VCM read registers for manual control with typical values

Register Address	Command	Typical Values for Signed 16b Register
256	software revision level	0x14BF = 5311 / 1000 = version 5.311
257	controller status	0x0003 = unknown meaning
258	faults	0x0000 = no faults 0x0020 = motor hall sensor fault 0x0100 = network communication timeout
259	controller temperature	0x0010 = 6 / 1 = 16 deg C
260	vehicle speed	0x0164 = 356 / 256 = 1.39 km/hr
261	motor temperature	0x000F = 15 / 1 = 15 deg C
262	motor current	0x0080 = 128 / 32 = 4 A
263	motor rpm	0x00E6 = 230 / 1 = 230 rpm
264	motor speed	0x0239 = 569 / 40.96 = 13.89 % speed
265	battery voltage	0x0636 = 1590 / 32 = 49.7 V
266	battery current	0xFFF5 = -11 / 32 = -0.34 A (probably incorrect)
267	battery state of charge	0x0064 = 100 / 1 = 100 %
268	battery power	0xFFF0 = -16 / 1 = -16 W (probably incorrect)
269	last fault	0x0000 = no faults
270	throttle voltage	0x21E9 = 8681 / 4096 = 2.12 V
271	brake 1 voltage	0x2C4A = 11338 / 4096 = 2.77V
272	brake 2 voltage	0x2164 = 8548 / 4096 = 2.07 V

Because hub motors are mounted facing in opposite directions, positive speed (0x0000 to 0x1000) for Rhub and negative speed (0x000 to 0xF000) for Lhub cause the vehicle to move forward. Similarly, negative speed for Rhub and positive speed for Lhub cause the vehicle to move in reverse. The VCM limited vehicle reverse speed to approximately 60% of maximum forward speed with Rhub values 0x0000 to 0xF680 and Lhub values 0x000 to 0x0980.

BAC1000s have a failsafe watchdog timer in case RS-485 communication is lost. When a BAC1000 is set to remote state = 2 = run, it will automatically shut down and indicate a fault if another valid command is not received within a specific amount of time called the command timeout threshold. The command timeout threshold must be set to zero using register 32 to disable this timer and allow continuous motor operation.

If a motor fault occurs, the error can be identified using command read faults 258. However, faults cannot be reset using Modbus commands. The LOGIC enable line must be toggled to reset faults and restart the controller.

Lastly, the VCM inserted a one-byte rollover counter into the upper byte of the value for register 493 remote state in Table 9.1 to help with diagnostics. The upper byte is ignored by BAC1000s and only the lower byte is used either as 0x01 = 1 = idle or 0x02 = 2 = run.

9.04 Manual Power Activation Sequence

This activation sequence is provided on page 84 of the shop manual [45]. All pin numbers and signal names refer to the 12-pin connector on the contactor with an 8-wire cable shown in Figure 9.2.

The VCM board receives 48 VDC from BATTERY+ (pin 1) and ground from RETURN (pin 6) of the contactor but VCM electronics are not powered until the KEY SWITCH is closed.

If manual sensors/switches are correctly activated (foot off throttle, brake applied, charger disconnected, seat switch closed, PTO off) and the Vehicle Start/Stop button on the front of the VCM is pressed, the VCM will begin activation of the contactor and motor controllers.

First, the VCM will provide 48 VDC back to PRE_CHARGE (pin 4) through a 100 Ω resistor. The PRE_CHARGE terminal bypasses the contactor relay and provides 48 VDC (albeit low current because of the 100 Ω resistor) directly to the motor power lines. This allows large capacitors inside the BAC1000s to charge slowly and will prevent high inrush of current when the contactor relay is closed. <u>Note that BAC1000 internal capacitors may stay charged for two to three minutes after power has been removed.</u>

Second, the VCM provides 48 VDC back to CONTACTOR+ (pin 2) and ground to CONTACTOR- (pin 3) across the contactor coil to activate the contactor relay.

Third, the VCM provides 48 VDC back to LOGIC (pin 5) which is passed to BAC1000s to enable logic circuits.

Lastly, the VCM begins transmitting RS-485 serial bus data to 485A (pin 12) and 485B (pin 11) which is daisy-chained to all four BAC100s. It should be noted that BAC1000s in remote state = 2 = run will shut down and indicate a fault if they do not receive another valid command within the command timeout threshold.

9.05 Cub Cadet Motor Shield

A custom microprocessor circuit called Cub Cadet motor shield (CCMS) was designed using an Arduino Mega to replace the VCM and control motor speed using standard RC signals.

A schematic is shown in Figure 9.5 and the corresponding printed circuit board (PCB) is shown in Figure 9.6. The CCMS provides optical isolation between the Mega and 48 VDC signals in the control panel. It connects to the control panel using the same 12 pin connector as the VCM. The PCB was designed as a shield that sits directly on top of the Mega.

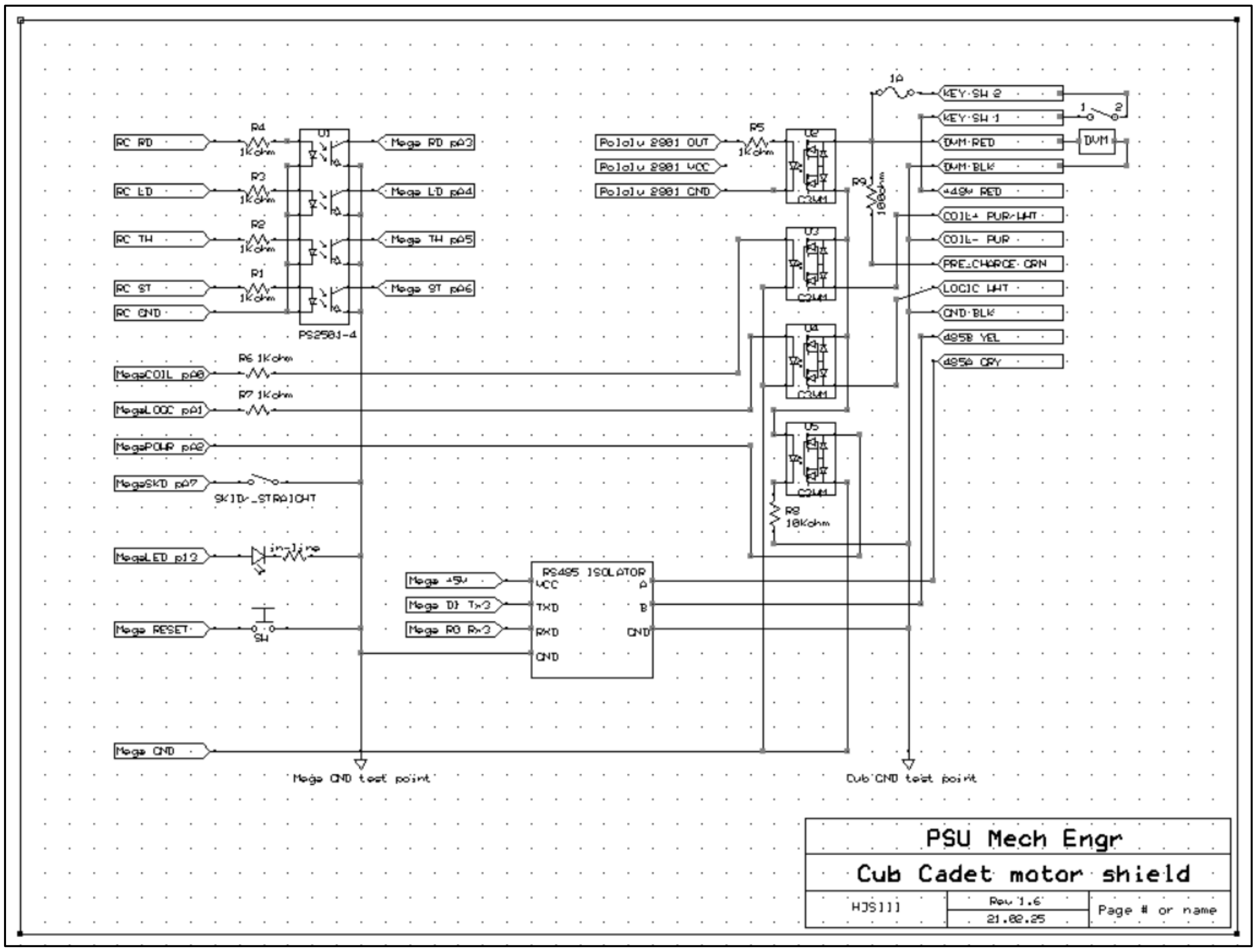


Figure 9.5 - Cub Cadet motor shield schematic

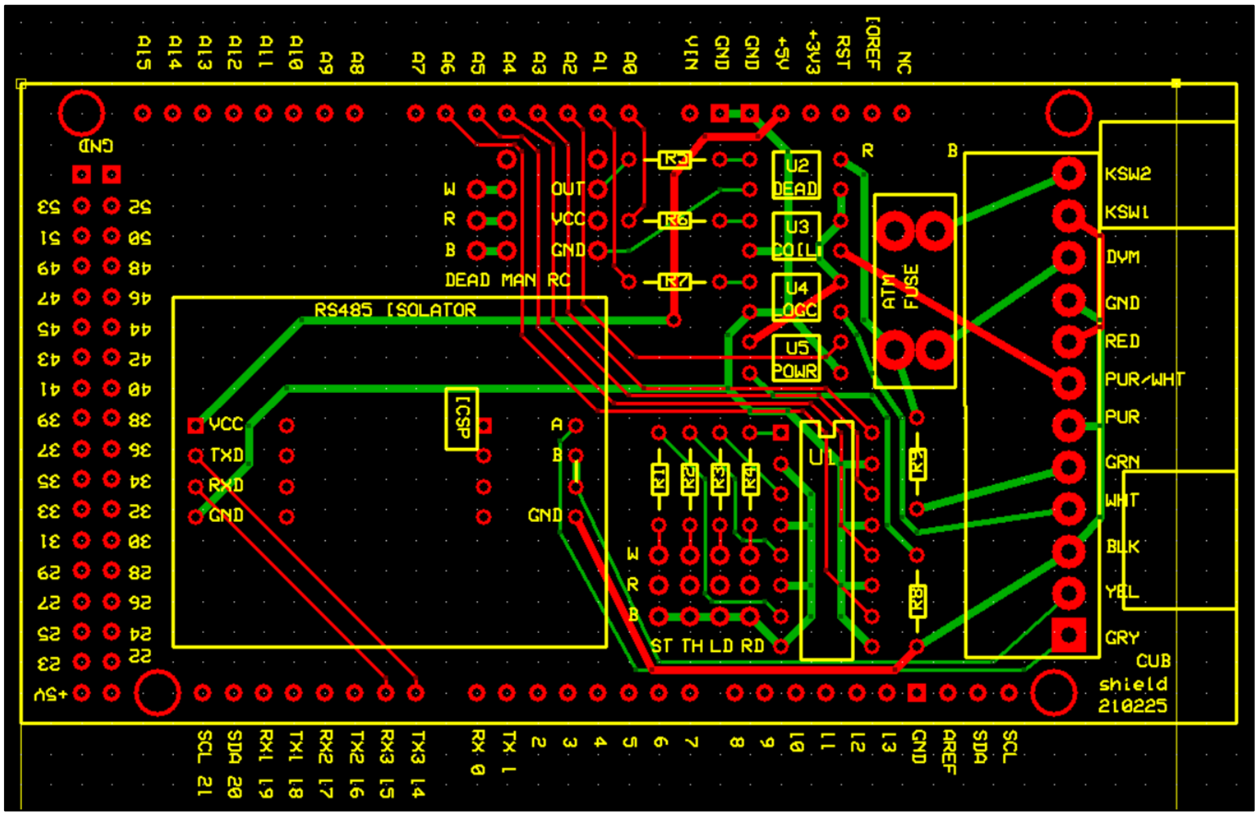


Figure 9.6 - Cub Cadet motor shield printed circuit board

Major components in the CCMS include a PS2501-4 optoisiolator U1, four Omrom G3VM-61A1 solid state relays (SSR) U3/U4/U5/U6, a Pololu 2801 RC switch, an RS-485 isolation module, an ATM fuse holder and a 12-pin terminal block for 16 AWG wire. Requisite external connections are listed in Table 9.3.

Table 9.3 - External connections to Cub Cadet motor shield

External Connections	*Location*
8 color coded wires from 12 pin connector on contactor	8 pin terminal block at right of PCB
key switch	8 pin terminal block at right of PCB
digital voltmeter (DVM)	8 pin terminal block at right of PCB
1 A fuse to protect coil	ATM fuse holder at upper right of PCB
Pololu 2801 RC switch	daughter board in upper center of PCB
dead man RC signal to Pololu 2801	3 pin header at left of Pololu 2801
four RC signals for steering, throttle, Ldeck, Rdeck	four 3 pin headers in lower center of PCB
dead man active LED indicator	pin 13 header
SKID/_STRAIGHT switch (toggle SPST NO)	pin A7 header
RESET switch (momentary SPST NO)	pin RST
+12 VDC Mega power	Mega power connector

The PS2501-4, Omron SSRs and RS-485 isolators provide full optical isolation between the Arduino Mega and the Cub Cadet control panel. Consequently, the Mega must have its own independent power supply.

The CCMS requires two independent RC systems for operation. Both RC systems must provide their own power for their respective receivers (Rx). The primary RC system with at least 4 channels will provide standard pulse width modulated (PWM) signals to control hub and deck motors. Signals for steering ST, throttle TH, left deck LD and right deck RD are required.

A second dead man RC system with at least one channel must be used for safety operation. The Pololu 2801 RC switch shown in Figure 9.7 reads the dead man RC signal and sets OUT based on pulse length. If there is no dead man RC input signal, OUT will be low and the yellow LED at the bottom right of the 2801 will blink at 50% duty cycle with period of 1 sec. If dead man RC pulses are less than 1.7 msec, OUT will be low and the LED be off with a brief blink on once per second. If dead man RC pulses are longer than 1.7 msec, OUT will be high and the LED will be on with a brief blink off once per second. The VCC-VRC jumper on the bottom of the Pololu 2801 must be soldered closed to allow the dead man Rx to power the 2801. The VCC pin from the 2801 must not be connected to the Mega +5 VDC pin.

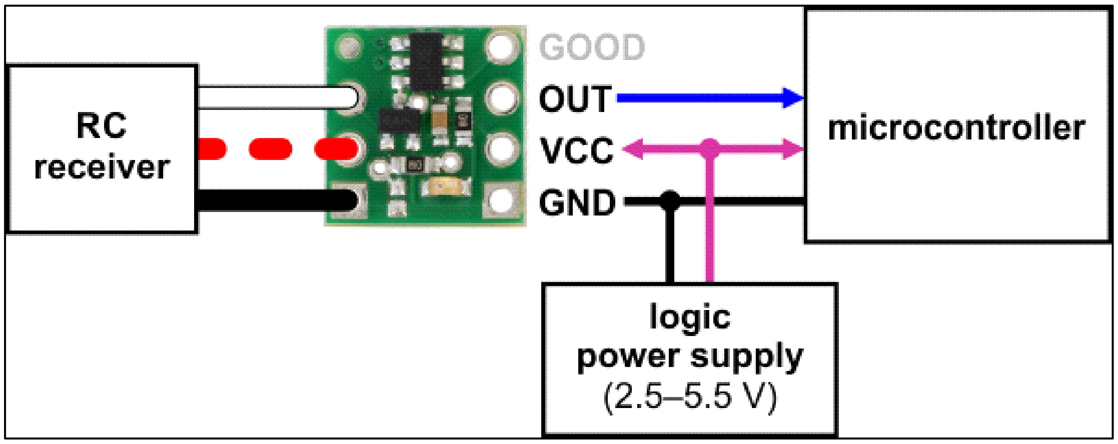


Figure 9.7 - Pololu 2801 dead man RC safety system

9.06 Microprocessor Power Activation Sequence

The CCMS receives 48 VDC from the contactor on the red +48V and black GND wires in the terminal block. However, BAC1000 electronics are not powered until an external key switch is closed between the top two pins of the terminal block. Closing the key switch provides power through the fuse to the DVM, the green PRE-CHARGE wire and Omron SSR U2.

Please note the description and warning about BAC1000 capacitor charging in Section 9.03 above. The DVM helps discharge BAC1000 capacitors.

A valid dead man RC signal with OUT set high will activate the dead man SSR U2 to provide 48 VDC to the other SSRs U3/U4/U5.

After checking power, the Mega will measure PWM center values for RC signals ST/TH/LD/RD and will not proceed unless valid signals are available. The Mega will then check the output of SSR U5 to see if 48 VDC is provided by the dead man SSR U2. The Mega will not proceed unless power is activated by the dead man safety circuit.

The Mega will then energize SSR U4 to connect 48 VDC to the white LOGIC wire which is passed to BAC1000s. This allows the Mega to begin transmitting RS-485 serial bus data over the 485A gray wire and the 485B yellow wire through the RS-485 isolator.

Lastly the Mega will energize SSR U3 to connect 48 VDC to the purple/white contactor COIL+ wire. The purple COIL- wire is internally connected to the black GND wire.

Note that this power sequence with LOGIC before COIL is slightly different from the manual power sequence. It allows the CCMS to interrogate BAC1000s via RS-485 before applying power though the contactor relay.

9.07 User Power-on Sequence

The recommended user activation sequence is provided in Table 9.4. However sufficient logic interlocks are provided to prevent major malfunctions if any components fail or are activated out of sequence. Loss of the RC dead man signal will cause all RZT-S motors to stop.

Table 9.4 - User Power-On Sequence

User Interface	*Indicator*
1) RC transmitter (Tx) for ST/TH/LD/RD	
2) RC receiver (Rx) for ST/TH/LD/RD	
3) Mega power	
4) key switch	DVM shows RZT-S battery voltage
5) RC transmitter (Tx) for dead man signal	
6) RC receiver (Rx) for dead man signal	Pololu 2801 LED
7) valid RC dead man signal	dead man LED indicator

9.08 Microprocessor Logic Flow

The Mega may be reset at any time by pushing a momentary SPST NO switch connecting the RST pin and Mega ground.

At startup, the Mega will measure PWM center values for RC signals ST/TH/LD/RD and will not proceed unless valid signals are available.

The Mega will then check the output of SSR U5 to see if 48 VDC is provided by the dead man SSR U2. The Mega will not proceed unless power is activated by the dead man safety circuit. The Mega will indicate if the dead man signal is active by illuminating an external LED using pin 13.

The Mega will then energize SSR U4 to connect 48 VDC to the white LOGIC wire to begin transmitting over the RS-485 serial data bus. The Mega will interrogate all four BAC1000s for faults, set command timeout threshold to zero for all four BAC1000s and set all four BAC1000s to idle. If no motor faults are detected, the Mega will energize SSR U3 to activate the coil.

Lastly, the Mega will check the logic level on pin A7 controlled by the SKID/_STRAIGHT switch (toggle SPST NO) to decide how to control the hub motors as described below.

The Mega then executes an infinite loop that reads four RC signals (ST, TH, LD and RD), sends motor speed commands to all four BAC1000s and performs several failsafe checks each time through the loop.

Standard RC PWM servo signals with pulses at 50 Hz must be used for ST, TH, LD and RD. Center values for all four channels are measured at startup.

The hub motors can be controlled using skid steer mixing where ST controls vehicle direction and TH controls vehicle forward/reverse speed. Alternately, the RC signals can be sent straight through to the hub motors where ST controls Lhub directly and TH controls Rhub directly.

Desired speeds for Lhub and Rhub are sent to respective BAC1000s. Note that negative Lhub speed values cause the vehicle to move forward and vice versa. Ldeck and Rdeck are operated at either 0 speed or 100% speed. All motors are operated at either 0 or 100% motoring and braking current.

For skid steer mixing, ST is mapped at 1 ms pulse duration for full left and 2 ms for full right. TH is mapped at 1 ms pulse duration for full reverse and 2ms for full forward.

For straight through motor control, ST is mapped at 1 ms pulse for Lhub full reverse and 2 ms for Lhub full forward. TH is mapped at 1 ms pulse for Rhub full reverse and 2 ms for Rhub full forward.

Left deck LD and right deck RD are mapped below 1.7 ms for off and above 1.7 ms for on.

Two skid steer algorithms were implemented to test for preferred operation. Both are based on the simple skid steer lookup table for a left wheel/track shown in Table 9.5. The lookup table for a right wheel/track is right/left symmetric. The first algorithm maps RC steering (-7 left <= ST <= 7 right) and RC throttle (-7 reverse <= TH <= 7 forward) and then simply does a lookup for left wheel/track output speed (-7 reverse < = left wheel/track < = 7 forward). The lookup value is then scaled -4096 to 4096 to send to a BAC1000. The second algorithm maps RC steering (-500 left <= ST <= 500 right) and RC throttle (-500 reverse <= TH <= 500 forward) and uses three linear functions to emulate the lookup table and compute left wheel/track speed output directly (-4000 reverse <= left wheel/track <= 4000 forward).

Table 9.5 - Skid steer lookup table for left wheel/track

				LEFT						STEERING						RIGHT	
			-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
	7		0	1	2	3	4	5	6	7	7	7	7	7	7	7	7
	6		-1	0	1	2	3	4	5	6	6	6	6	6	6	6	7
FORWARD	5		-2	-1	0	1	2	3	4	5	5	5	5	5	5	6	7
	4		-3	-2	-1	0	1	2	3	4	4	4	4	4	5	6	7
	3		-4	-3	-2	-1	0	1	2	3	3	3	3	4	5	6	7
	2		-5	-4	-3	-2	-1	0	1	2	2	2	3	4	5	6	7
	1		-6	-5	-4	-3	-2	-1	0	1	1	2	3	4	5	6	7
THROTTLE	0		-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
	-1		-7	-6	-5	-4	-3	-2	-1	-1	0	1	2	3	4	5	6
	-2		-7	-6	-5	-4	-3	-2	-2	-2	-1	0	1	2	3	4	5
	-3		-7	-6	-5	-4	-3	-3	-3	-3	-2	-1	0	1	2	3	4
	-4		-7	-6	-5	-4	-4	-4	-4	-4	-3	-2	-1	0	1	2	3
REVERSE	-5		-7	-6	-5	-5	-5	-5	-5	-5	-4	-3	-2	-1	0	1	2
	-6		-7	-6	-6	-6	-6	-6	-6	-6	-5	-4	-3	-2	-1	0	1
	-7		-7	-7	-7	-7	-7	-7	-7	-7	-6	-5	-4	-3	-2	-1	0

Three primary subroutines were developed to write one BAC1000 register for setting parameters, to write four contiguous registers for motor control and to read multiple contiguous registers. The four contiguous registers for motor control are the same as VCM manual control shown in Table 9.1 above. Individual registers for reading/writing parameters are listed in Table 9.6 below.

Table 9.6 - BAC1000 registers for CCMS operation with typical values

Register Address	*Command*	*Typical Values for Signed 16b Register*
32	command time out threshold (read/write)	0x0000 = disable timer for continuous operation 0x0100 = 256 / 1 = 256 msec (default)
258	faults (read only)	0x0000 = no faults 0x0020 = motor hall sensor fault 0x0100 = network communication timeout
259	controller temperature (read only)	0x0010 = 6 / 1 = 16 deg C
260	vehicle speed (read only)	0x0164 = 356 / 256 = 1.39 km/hr
261	motor temperature (read only)	0x000F = 15 / 1 = 15 deg C
263	motor rpm (read only)	0x00E6 = 230 / 1 = 230 rpm
264	motor speed (read only)	0x0239 = 569 / 40.96 = 13.89 % speed
265	battery voltage (read only)	0x0636 = 1590 / 32 = 49.7 V

For each cycle within the infinite loop, desired remote speed is written to each BAC1000 using register 490. Failsafe checks are then performed including motor faults using register 258, valid RC signals (ST, TH, LD and RD), output of SSR U5 indicating valid dead man RC signal and actual motor speed using register 264. Actual motor speed less than 50 percent of desired remote speed is a simple test for motor overload (e.g. vehicle collision, manual brake applied, deck motors clog). If any check fails, all motors shut down and the Mega will restart following the logic flow described above.

Future code within the infinite loop could be developed to provide motor acceleration or deceleration for sudden changes in commanded speed. This could be implemented by ramping speed over several cycles of the infinite loop or by using lower motoring current. Performance could be enhanced by experimenting with both motoring and braking current. Future upgrades could also check motor and controller temperatures to prevent overheating, vehicle speed in kilometers per hour or miles per hour for display purposes and battery voltage to prevent battery discharge problems.

9.09 Future Control Concepts

The CCMS described above provides remote operation of an RZT-S using RC signals. Alternately a radio modem could be used to send/receive serial communications directly between a base station and RZT-S without any RC.

One approach for the remote serial data could be to send/receive actual BAC1000 Modbus commands where the shield acts only as a pass-through device. A second approach for the remote serial data would be to adopt/develop a standardized vehicle control language where the shield interprets vehicle control commands into BAC1000 commands.

A radio modem would also allow the RZT-S to send local sensor information back to the base station for enhanced collision avoidance path planning.

Chapter 10

Retrofitting the Cub Cadet RZT-S Zero

10.01 Overview

This chapter describes the modifications made to the factory-built Cub Cadet RZT-S Zero to adapt it for autonomous control. The steering system was altered so that mower's speed and heading could be controlled with the rear drive wheels. An autonomy platform was constructed to mount the electronics and sensors.

10.02 Simplifying Control of the Cub Cadet

The Cub Cade RZT-S Zero is designed to be manually steered by an operator using a steering wheel, as shown in Figure 10.1. Most zero-turn mowers are controlled by the operator with two lever inputs, each one controlling the speed and direction of its respective drive wheel. In its unmodified configuration, the Cub Cadet detects the steer angle input by the operator and adjusts the speed differential of the drive wheels. While this Ackermann steering design makes steering more intuitive for the operator, it complicates unmanned control of the vehicle. Since speed and heading can be controlled by the speed and direction of the independent drive wheels, a mechanical steering system is unnecessary. The Pixhawk system developed for UGV01 can be ported to the Cub Cadet, but only if it is a differentially steered vehicle. The Cub Cadet was altered so that it could be controlled entirely by the speed of its left and right wheels. Therefore, it was altered so that it could be controlled by the 2-channel output from the Pixhawk.

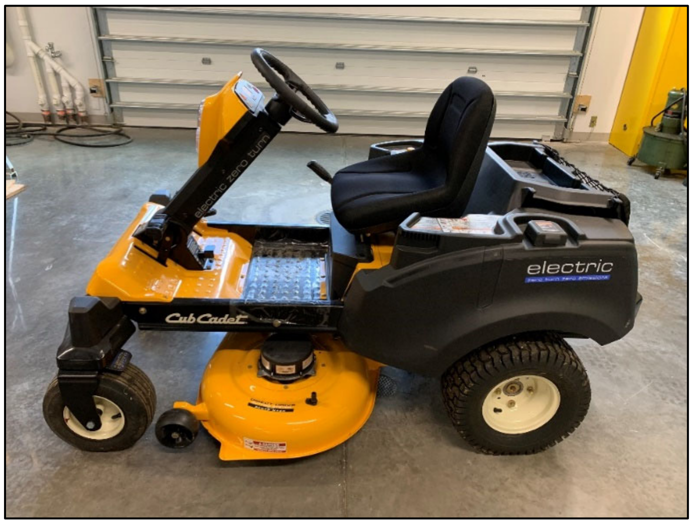


Figure 10.1 - Unmodified Cub Cadet RZT-S Zero

The front wheel axles are carried by yokes. The steering wheel on the unmodified Cub Cadet rotates each yoke via a steering gear mounted to the base of the pivot shaft, shown in Figure 10.2. Removing the steering gear from the steering system allowed it to spin freely. However, the original yoke does not have a caster offset. A caster design offsets the wheel axle behind the pivot shaft of the yoke. With the pivot shaft ahead of the wheel's contact patch, the heading of the wheel will follow the heading of the pivot shaft and therefore the heading of the vehicle. Without a caster offset, the free-spinning yokes on the Cub Cadet RZT-S will not follow the heading of the mower, as shown in Figure 10.2.

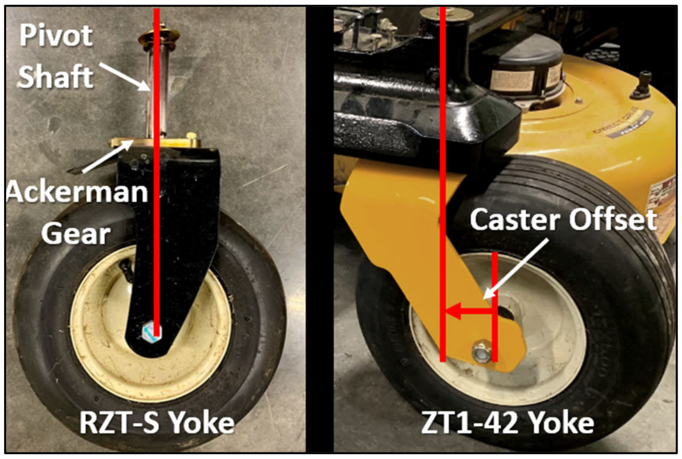


Figure 10.2 - Comparison of the Cub Cadet RZT-S and ZT1-42 wheel yokes

Other Cub Cadet zero-turn mower models have free spinning caster yokes. The Cub Cadet ZT1-42 is one of these models. Its yokes use the same axle bolt as the RZT-S. Therefore, the wheel and tire from the RZT-S fit on the ZT1-42 caster yoke. Additionally, the dimensions of the pivot shaft on the ZT1-42 yoke match those of the RZT-S yoke. Two ZT1-42 caster yokes were purchased from Cub Cadet to replace the original RZT-S yokes.

The steering gear on the original yokes also acted as thrust bushings. Since the steering gear was not needed on the caster yoke, a thrust bushing of the same thickness (10 millimeters) was needed. Two thrust bushings were cut from a 10-millimeter-thick steel plate and installed on the caster yokes, as shown in Figure 10.3.

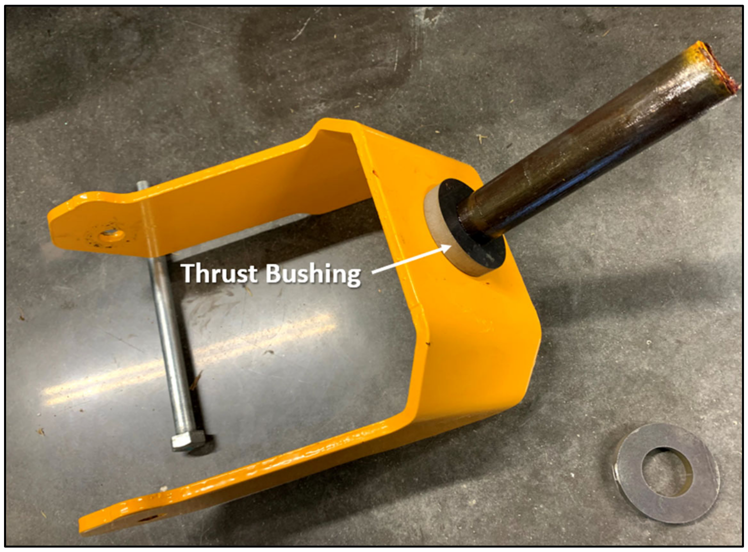


Figure 10.3 - Thrust bushing installed on a caster yoke

Installing caster yokes on the Cub Cadet offset the wheel and tire farther from the pivot shaft, as shown in Figure 10.4. As a result, the spin radius of the left-side tire caused interference with the anti-scalping wheel on the mowing deck.

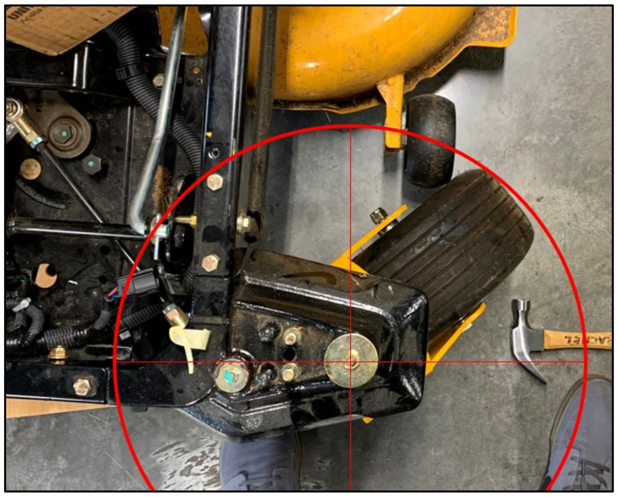


Figure 10.4 - Increased tire spin radius caused by the new caster yoke

To allow the left caster to spin freely, the deck wheel was removed and relocated outside of the spin radius. Relocation of the deck wheel required fabrication of a deck mounting block.

The mounting block was first 3D printed to assess fit and then waterjet cut from a 3.5x12x2-inch block of aluminum as shown in Figure 10.5.

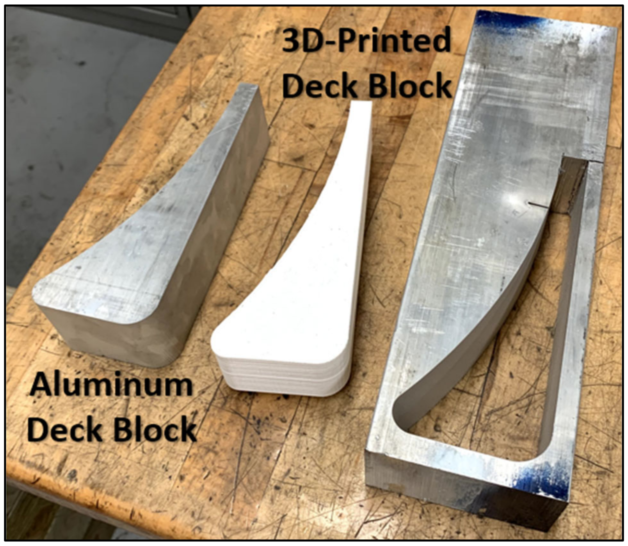


Figure 10.5 - Fabrication of deck mounting block

The aluminum deck block was mounted to the mowing deck with 3-inch long, 6-millimeter carriage bolts. Two holes were bored through the deck block and deck for the bolts. Two additional holes were drilled and tapped in the side of the block for the attachment on a 12-inch long by 1/16-inch-thick steel C channel bar. A hole was bored in the end of the steel bar to mount the deck wheel.

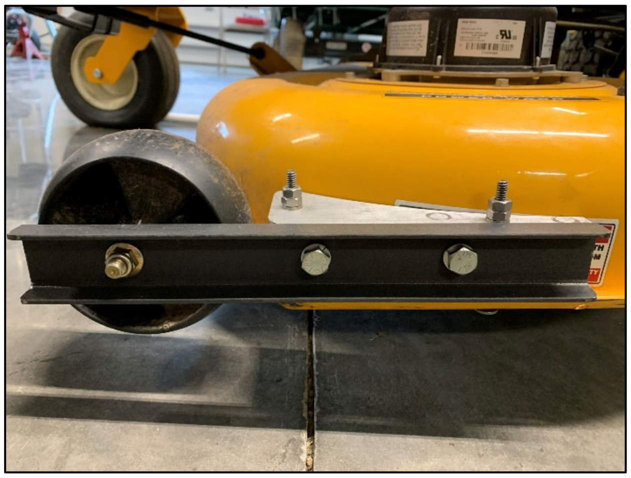


Figure 10.6 – Relocated mowing deck wheel with custom-fabricated mount

Relocating the mower deck wheel eliminated interference with the left caster yoke, as shown in Figure 10.7. With free-spinning front casters, the Cub Cadet could be controlled as a differential steer vehicle.

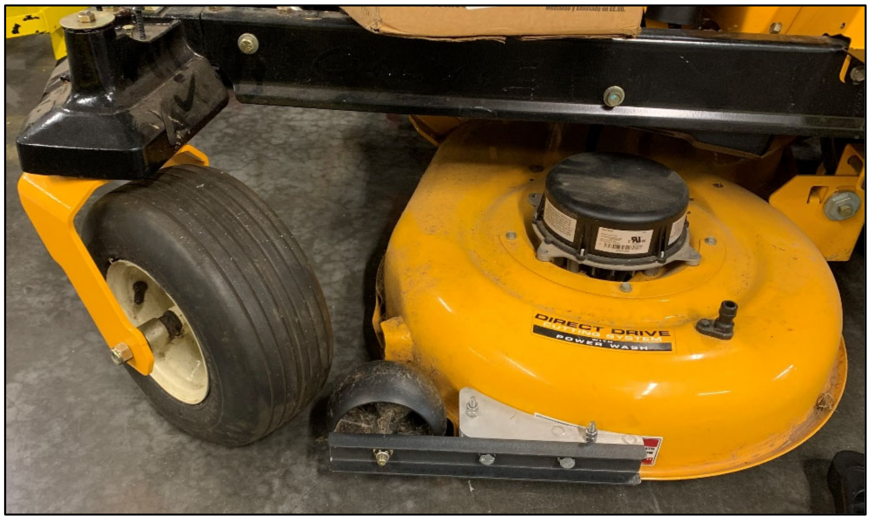


Figure 10.7 - Relocated deck wheel outside sufficiently far from the caster yoke spin radius

10.03 Integrating the Autonomous Control System

The Cub Cadet was further modified to accommodate integration of autonomy hardware components. The sensitive electronics and signal wires of the Pixhawk system had to be located away from the interference caused by high-current wires and motors. The sensor of primary concern is the magnetometer, which is affected by ferrous metal.

The Cub Cadet steering column and seat were removed. Neither of these features are needed for autonomous control. Beneath the seat is the flat deck shown in Figure 10.8.

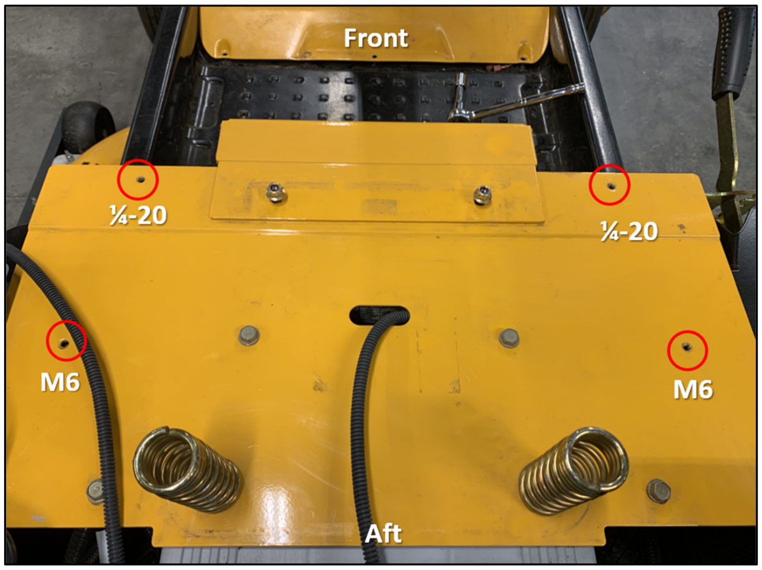


Figure 10.8 - Flat deck below seat used for mounting the autonomy platform

In Figure 10.8, four existing threaded bolt holes are circled. Each hole is labeled with its thread type and pitch. The unmodified Cub Cadet used these holes to mount plastics. Two ¼-20-2-inch bolts and two M6-50-millimeter bolts were used to mount the autonomy platform. First, two blocks of 2x4 inch board were mounted to the seat deck. Next, a 12x24 inch sheet of OSB plywood was mounted to the wooden blocks, as shown in Figure 10.9. A 24x24x12 inch cage was built from 1x1 inch aluminum tubing with press-fit corners. This cage was bolted to the plywood affixed to the mower, as shown in Figure 10.9.

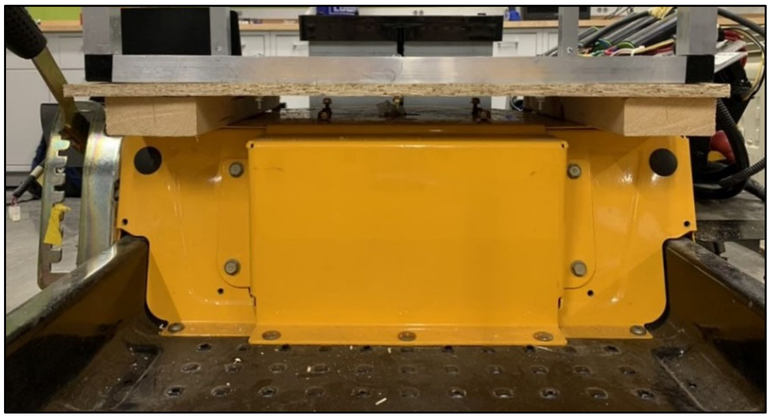


Figure 10.9 – Wooden base of the autonomy platform on the seat deck

An additional sheet of 12x24 inch OSB plywood was mounted to the top of the cage. The resultant autonomy platform shown in Figure 10.10 provides ample space for hardware. A minimal amount of ferrous metal is present in the autonomy platform. Wood was selected as the material of the mounting surface because it is easy to add and remove hardware during prototyping. The autonomous orchard mower system can be expected to undergo several iterations of design changes.

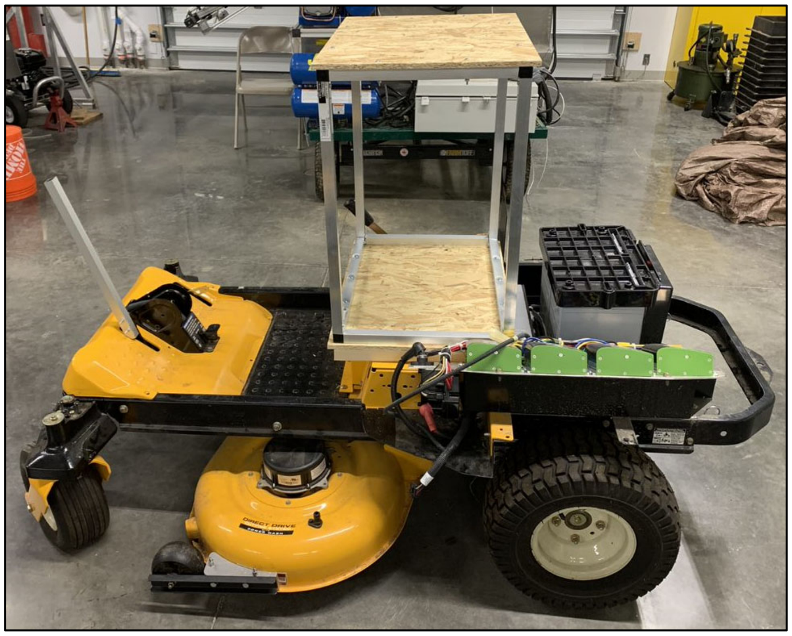


Figure 10.10 - Cub Cadet retrofitted with autonomy platform

Removing the seat from the Cub Cadet made it difficult to use the brake pedal. Although the mechanical parking brake is not controlled by the autonomous system, it is an important safety feature. An emergency parking brake lever was fabricated from a 1x1 inch aluminum bar. The bar was shaped and bolted to the pedal bracket, as shown in Figure 10.11.

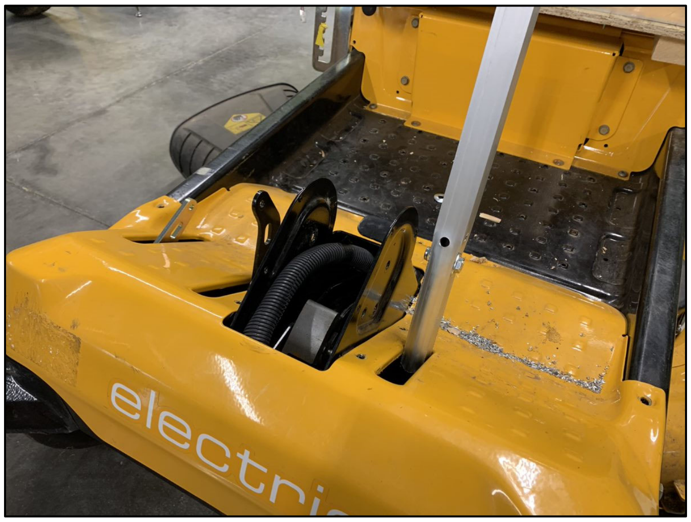


Figure 10.11 - Brake lever bolted to the existing pedal bracket

The entire Pixhawk system from UGV01 was removed and installed onto the top of the autonomy platform. The carbon-rod compass pedestal was mounted at the front of the platform, along the center line. The GNSS antenna was mounted at the aft, also along the centerline. Constrained by the length of the compass signal wire, the electronics housing was mounted between the antenna and compass, as shown in Figure 10.12. As on UGV01, an 11.7V LiPo battery was used to power the system.

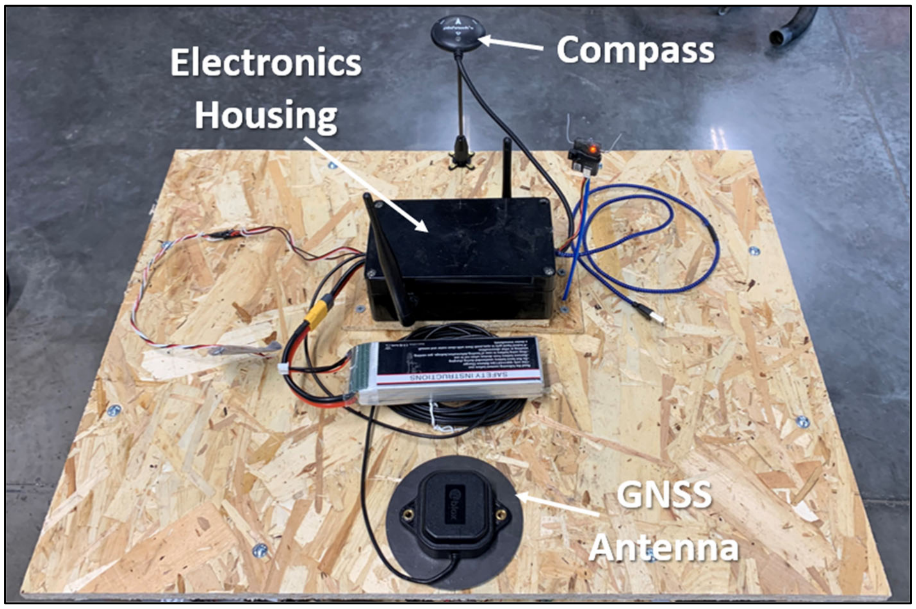


Figure 10.12 - Pixhawk system mounted on the Cub Cadet autonomy platform

The Pixhawk was booted and then connected to Mission Planner to ensure the autonomy platform did not interfere with the magnetometer. The true heading of the Cub Cadet was known to be about 36 degrees and the magnetometer read a heading of about 49 degrees. This test verified that the magnetometer was functional, but it needed recalibration. Recalibration will be done during field tests at Rock Springs Orchard.

The Cub Cadet Motor Shield (CCMS) and Arduino Mega described in Section 9.05 were housed in a control box. The user interface of this control box is detailed in Table 9.4. The CCMS control box takes in five R/C channels, as shown in Figure 10.13 and Figure 10.14.

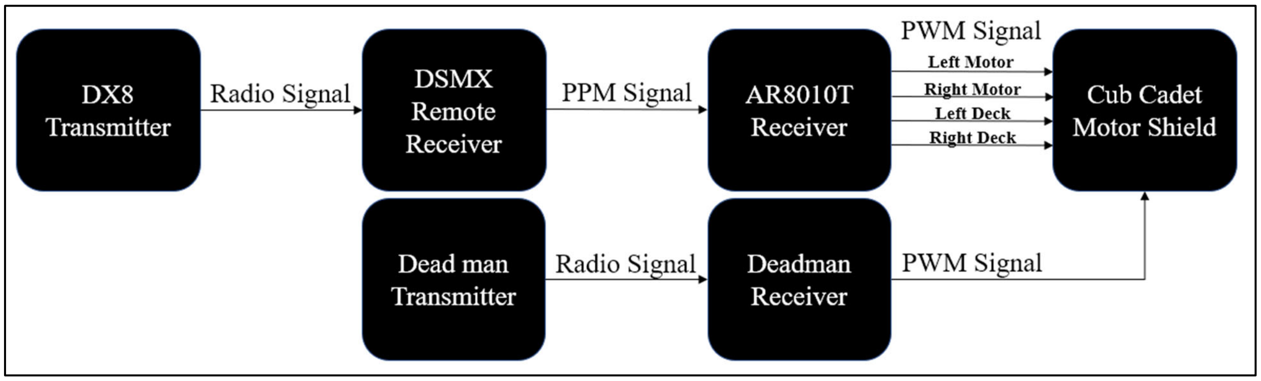


Figure 10.13 - Signal flow of R/C CCMS system

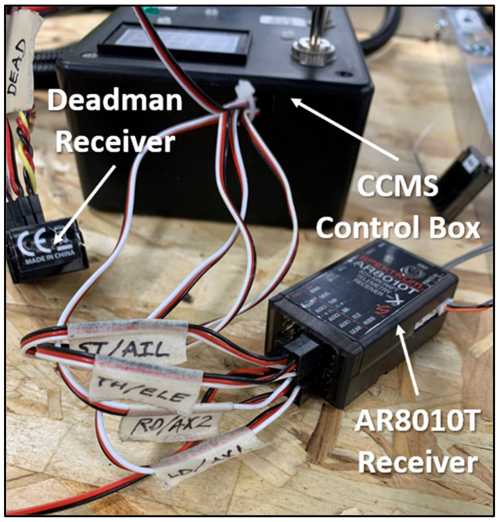


Figure 10.14 - R/C receivers wired to the CCMS control box

The CCMS accepts five PWM input signals, four of which the Pixhawk system had to be able to modulate. The fifth PWM channel is for the dead man. Similar to integrating the Pixhawk into the R/C system UGV01, the AR8010T receiver is replaced by the Pixhawk, as shown in Figure 10.15 and Figure 10.16.

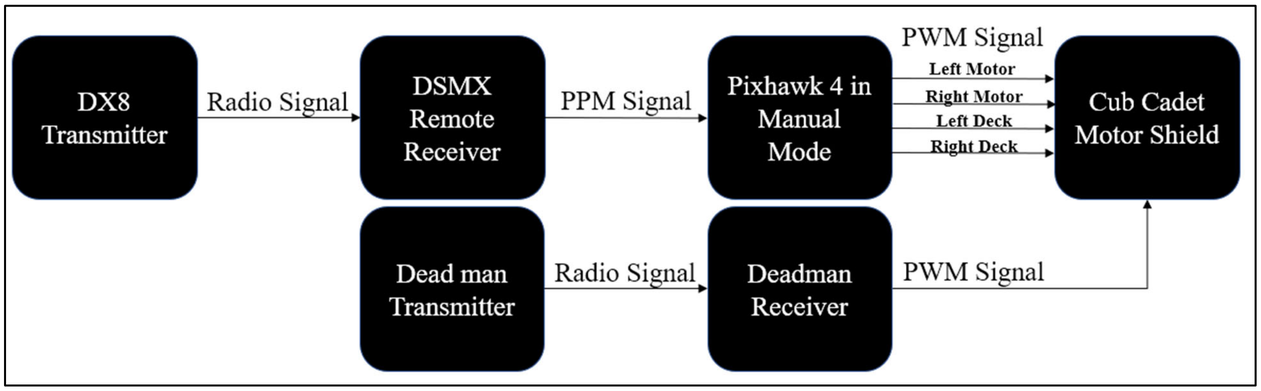


Figure 10.15 - Signal flow of Pixhawk CCMS system in manual mode

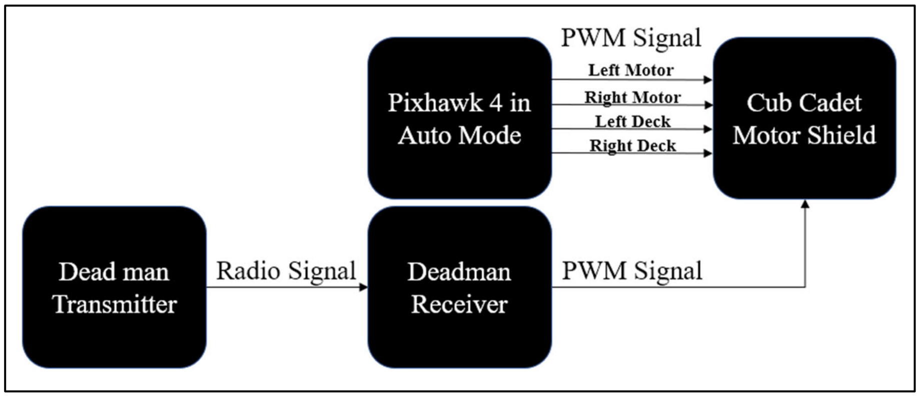


Figure 10.16 - Signal flow of Pixhawk CCMS system in auto mode

The Pixhawk system taken from UGV01 had two PWM output signals: one for the left motor and one for the right. Two additional output channels were configured in ArduPilot for the left and right mower deck motors. Servo channels 4 and 5 were assigned to R/C channels 5 and 7, respectively. The Learn Cruise function was remapped to the left stick. The ArduPilot parameter adjustments are detailed in Table 10.1.

Table 10.1 – ArduPilot parameter adjustments for Pixhawk CCMS system

Parameter	*Value*	*Function*
SERVO4_FUNCTION	55	Map R/C channel 5 (switch A) to servo 4
SERVO5_FUNCTION	57	Map R/C channel 7 (switch F) to servo 5
RC5_OPTION	0	Remove Learn Cruise function from channel 5 (switch A)
RC1_OPTION	50	Add Learn Cruise function to channel 1 (left stick)

The Pixhawk was wired to the CCMS by connecting its PWM output channels to the proper CCMS inputs. The wiring connections of the R/C and Pixhawk CCMS systems are summarized in Table 10.2. The R/C configuration uses the CCMS in mixed-signal mode whereas the Pixhawk uses the CCMS in unmixed (straight) mode. In mixed mode, CCMS input ST controls steering and TH controls throttle. In straight mode, ST controls the left hub and TH the right hub.

Table 10.2 - Wiring connections of the CCMS systems

CCMS Input	*AR8010T Output*	*Pixhawk Output*	*Function*
ST	AIL	CH1	Steering/Left Hub
TH	ELE	CH3	Throttle/Right Hub
LD	AUX1	CH4	Left Deck
RD	AUX2	CH5	Right Deck

The Pixhawk was powered on and armed with the Cub Cadet drive wheels elevated off the ground, as shown in Figure 10.17. In manual mode, the Pixhawk successfully controlled the left and right hubs, as well as the right deck motor. As noted in Section 9.02, the left deck motor on the old RZT-S has a Hall sensor fault and therefore could not be operated. Testing of auto mode and tuning of performance is needed. This will be done at Rock Springs orchard in April 2021.

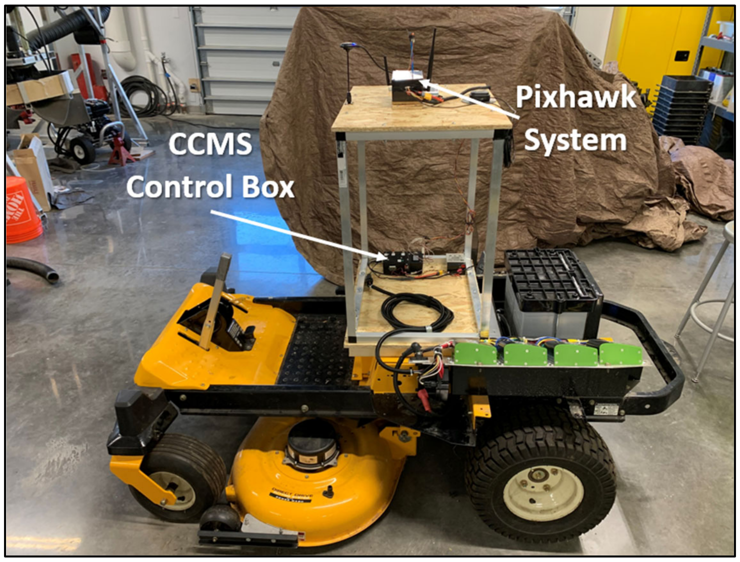


Figure 10.17 - Cub Cadet with Pixhawk system installed

Chapter 11

Future Work

The Cub Cadet with Pixhawk system will be field tested at Rock Springs Orchard in April 2021. Field testing during the course of the project was difficult due to a large amount of snowfall. Preliminary testing will be done on the test track. The same mission that was given to UGV01 will be given to the Cub Cadet. Based on the ability of the Pixhawk to control the Cub Cadet, parameters will be tuned to improve performance. The Cub Cadet will react slightly differently to the outputs from the Pixhawk since it pivots about the center of the rear axle, as opposed to UGV01 which pivots about its geometric center. After tuning performance, the Cub Cadet will be given orchard row missions in blocks A2 and A3, where the orchard rows are relatively wide. This will provide a higher tolerance of deviation to increase safety during testing.

The presence of foliage on trees can impact signal strength. UGV01 signal strength within orchard rows was assessed when the trees were bare. Satellite signal strength will be reassessed when the trees bloom in April 2021.

The next step in developing the control system for the Cub Cadet is integrating an onboard computer running ROS. The onboard computer will be a Raspberry Pi 3. Adding a companion computer to the Pixhawk will increase the local computational power available on the vehicle. ROS onboard allows for the use of more complex sensing tools, such as LiDAR and computer vision.

An array of ultrasonic sensors will also be added to the Cub Cadet for proximity sensing. The array will consist of 5 sensors mounted at the front of the vehicle: one facing forward, two at 15 degrees, and two at 30 degrees, as shown in Figure 11.1. This array will find the distance to objects directly in front of the vehicle for safety and object avoidance. It will also find the distance to the trees on either side of the vehicle. This information can be used to adjust the position of the mower between the rows of trees, correcting for deviation from the center.

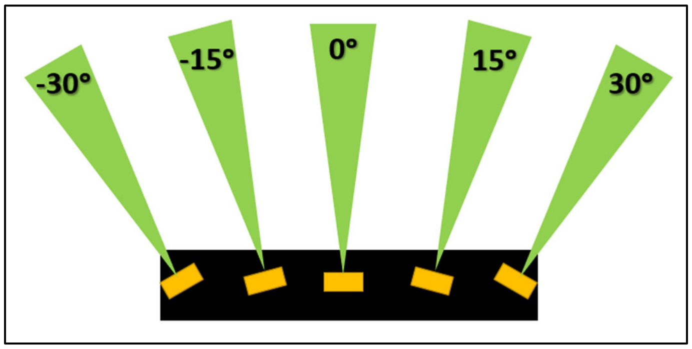


Figure 11.1 - Ultrasonic array design for the Cub Cadet

The future autonomous control system for the Cub Cadet will not use the Pixhawk autopilot controller. Rather, the onboard computer running ROS will perform the waypoint navigation tasks currently done by the Pixhawk. A package will be built for ROS to use satellite and sensor information for localization within the orchard. Mission plans will be given to the onboard ROS system via the ROS master ground control station.

The mission updating feature developed for ROS will be improved. In its current form, the user must know where the ground vehicle is within the orchard to generate a new mission. The new mission must guide the ground vehicle out of the row and create an obstacle-free path to the start of the new mission. To further automate the mission update sequence, a ROS-enabled computer could use its knowledge of the orchard (i.e. the locations of the tree rows) to generate an obstacle-free path to the start of the new mission. With the aid of range-finding sensors, the Cub Cadet will be better equipped to assist the path-finding algorithm. Specifically, it will be able to find its way out of an orchard row and around any obstacles between the end of the row and the start of the new mission's first row.

A new Cub Cadet RZT-S mower will be modified in the same manner as the old Cub Cadet described in Chapter 10. The new Cub Cadet does not have a Hall sensor fault on the left deck motor. Therefore, it is better equipped to mow orchard rows once the autonomy system has been tested and proved on the old Cub Cadet. A comparison of the old and new Cub Cadet mowers is shown in Figure 11.2.

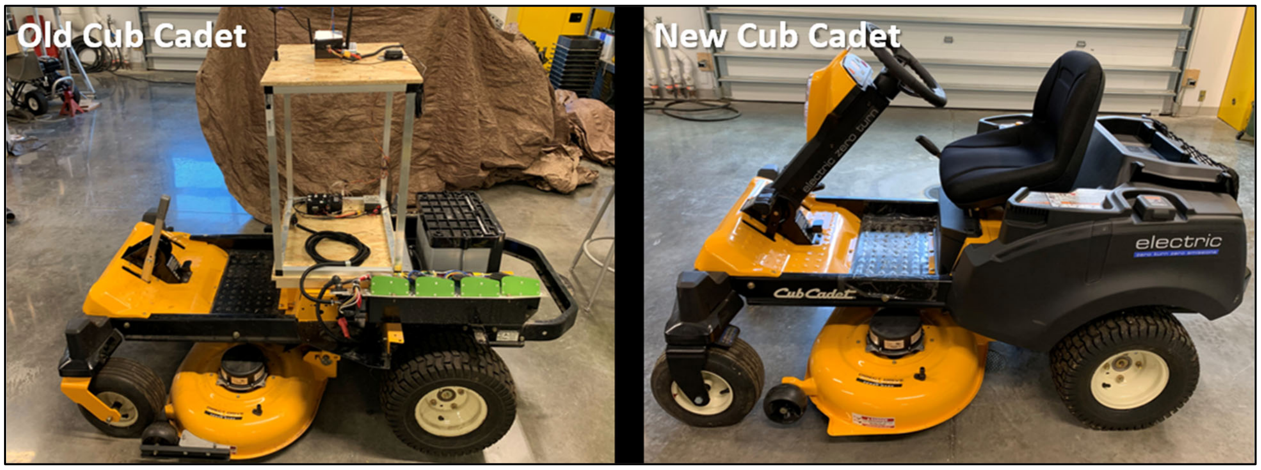


Figure 11.2 - Modified old Cub Cadet compared to the new Cub Cadet

Appendix A

Mission Planner Parameter Lists

The parameter files presented in this appendix have been generated from Mission Planner. The 919 parameters listed are for the ArduPilot Rover 4.0.0 firmware.

A.1 Non-RTK Configuration

In this configuration, the basic Pixhawk setup is used. The OEM GPS puck is used for localization. To convert the text below into a file that can be uploaded into Mission Planner, a continuous list of the parameter names with parameter values should be made in a text editor. This list of comma-separated parameter and value pairs should be saved with a *.param file extension. For example, this list of parameters in a text document can be saved as Basic_Pixhawk.param.

Index	*Parameter*	*Value*	*Index*	*Parameter*	*Value*
1	ACRO_TURN_RATE	180	**461**	RC11_MIN	1000
2	AHRS_COMP_BETA	0.1	**462**	RC11_OPTION	0
3	AHRS_CUSTOM_PIT	0	**463**	RC11_REVERSED	0
4	AHRS_CUSTOM_ROLL	0	**464**	RC11_TRIM	1500
5	AHRS_CUSTOM_YAW	0	**465**	RC12_DZ	0
6	AHRS_EKF_TYPE	2	**466**	RC12_MAX	2000
7	AHRS_GPS_GAIN	1	**467**	RC12_MIN	1000
8	AHRS_GPS_MINSATS	6	**468**	RC12_OPTION	0
9	AHRS_GPS_USE	1	**469**	RC12_REVERSED	0
10	AHRS_ORIENTATION	0	**470**	RC12_TRIM	1500
11	AHRS_RP_P	0.2	**471**	RC13_DZ	0
12	AHRS_TRIM_X	-0.01955	**472**	RC13_MAX	2000
13	AHRS_TRIM_Y	0.027536	**473**	RC13_MIN	1000
14	AHRS_TRIM_Z	0	**474**	RC13_OPTION	0
15	AHRS_WIND_MAX	0	**475**	RC13_REVERSED	0
16	AHRS_YAW_P	0.2	**476**	RC13_TRIM	1500
17	ARMING_ACCTHRESH	0.75	**477**	RC14_DZ	0

18	ARMING_CHECK	1	**478**	RC14_MAX	2000
19	ARMING_MIS_ITEMS	0	**479**	RC14_MIN	1000
20	ARMING_REQUIRE	1	**480**	RC14_OPTION	0
21	ARMING_RUDDER	2	**481**	RC14_REVERSED	0
22	ARSPD_TYPE	0	**482**	RC14_TRIM	1500
23	ATC_ACCEL_MAX	0.3	**483**	RC15_DZ	0
24	ATC_BAL_D	0.03	**484**	RC15_MAX	1900
25	ATC_BAL_FF	0	**485**	RC15_MIN	1100
26	ATC_BAL_FLTD	0	**486**	RC15_OPTION	0
27	ATC_BAL_FLTE	10	**487**	RC15_REVERSED	0
28	ATC_BAL_FLTT	0	**488**	RC15_TRIM	1500
29	ATC_BAL_I	1.5	**489**	RC16_DZ	0
30	ATC_BAL_IMAX	1	**490**	RC16_MAX	1900
31	ATC_BAL_P	1.8	**491**	RC16_MIN	1100
32	ATC_BAL_SPD_FF	1	**492**	RC16_OPTION	0
33	ATC_BRAKE	0	**493**	RC16_REVERSED	0
34	ATC_DECEL_MAX	0	**494**	RC16_TRIM	1500
35	ATC_SAIL_D	0	**495**	RC2_DZ	30
36	ATC_SAIL_FF	0	**496**	RC2_MAX	1898
37	ATC_SAIL_FLTD	0	**497**	RC2_MIN	1098
38	ATC_SAIL_FLTE	10	**498**	RC2_OPTION	0
39	ATC_SAIL_FLTT	0	**499**	RC2_REVERSED	0
40	ATC_SAIL_I	0.1	**500**	RC2_TRIM	1498
41	ATC_SAIL_IMAX	1	**501**	RC3_DZ	0
42	ATC_SAIL_P	1	**502**	RC3_MAX	1900
43	ATC_SPEED_D	0	**503**	RC3_MIN	1100
44	ATC_SPEED_FF	0	**504**	RC3_OPTION	0
45	ATC_SPEED_FLTD	0	**505**	RC3_REVERSED	0
46	ATC_SPEED_FLTE	10	**506**	RC3_TRIM	1500
47	ATC_SPEED_FLTT	0	**507**	RC4_DZ	30
48	ATC_SPEED_I	0.2	**508**	RC4_MAX	1902
49	ATC_SPEED_IMAX	1	**509**	RC4_MIN	1102
50	ATC_SPEED_P	0.2	**510**	RC4_OPTION	0
51	ATC_STOP_SPEED	0.1	**511**	RC4_REVERSED	1
52	ATC_STR_ACC_MAX	180	**512**	RC4_TRIM	1502
53	ATC_STR_ANG_P	2.5	**513**	RC5_DZ	0
54	ATC_STR_RAT_D	0	**514**	RC5_MAX	1901
55	ATC_STR_RAT_FF	0.2	**515**	RC5_MIN	1099
56	ATC_STR_RAT_FLTD	0	**516**	RC5_OPTION	50
57	ATC_STR_RAT_FLTE	10	**517**	RC5_REVERSED	0
58	ATC_STR_RAT_FLTT	0	**518**	RC5_TRIM	1500
59	ATC_STR_RAT_I	0.2	**519**	RC6_DZ	0

60	ATC_STR_RAT_IMAX	0.8	**520**	RC6_MAX	1901
61	ATC_STR_RAT_MAX	360	**521**	RC6_MIN	1099
62	ATC_STR_RAT_P	0.1	**522**	RC6_OPTION	0
63	AUTO_KICKSTART	0	**523**	RC6_REVERSED	0
64	AUTO_TRIGGER_PIN	-1	**524**	RC6_TRIM	1500
65	AVOID_ANGLE_MAX	1000	**525**	RC7_DZ	0
66	AVOID_BEHAVE	1	**526**	RC7_MAX	1901
67	AVOID_DIST_MAX	5	**527**	RC7_MIN	1099
68	AVOID_ENABLE	3	**528**	RC7_OPTION	0
69	AVOID_MARGIN	2	**529**	RC7_REVERSED	0
70	BAL_PITCH_MAX	2	**530**	RC7_TRIM	1500
71	BAL_PITCH_TRIM	0	**531**	RC8_DZ	0
72	BATT_MONITOR	0	**532**	RC8_MAX	1901
73	BATT2_MONITOR	0	**533**	RC8_MIN	1099
74	BATT3_MONITOR	0	**534**	RC8_OPTION	41
75	BATT4_MONITOR	0	**535**	RC8_REVERSED	0
76	BATT5_MONITOR	0	**536**	RC8_TRIM	1500
77	BATT6_MONITOR	0	**537**	RC9_DZ	0
78	BATT7_MONITOR	0	**538**	RC9_MAX	2000
79	BATT8_MONITOR	0	**539**	RC9_MIN	1000
80	BATT9_MONITOR	0	**540**	RC9_OPTION	0
81	BCN_ALT	0	**541**	RC9_REVERSED	0
82	BCN_LATITUDE	0	**542**	RC9_TRIM	1500
83	BCN_LONGITUDE	0	**543**	RCMAP_PITCH	1
84	BCN_ORIENT_YAW	0	**544**	RCMAP_ROLL	2
85	BCN_TYPE	0	**545**	RCMAP_THROTTLE	3
86	BRD_BOOT_DELAY	0	**546**	RCMAP_YAW	4
87	BRD_IO_ENABLE	1	**547**	RELAY_DEFAULT	0
88	BRD_OPTIONS	1	**548**	RELAY_PIN	-1
89	BRD_PWM_COUNT	8	**549**	RELAY_PIN2	-1
90	BRD_RTC_TYPES	1	**550**	RELAY_PIN3	-1
91	BRD_RTC_TZ_MIN	0	**551**	RELAY_PIN4	-1
92	BRD_SAFETY_MASK	0	**552**	RELAY_PIN5	-1
93	BRD_SAFETYENABLE	1	**553**	RELAY_PIN6	-1
94	BRD_SAFETYOPTION	7	**554**	RNGFND1_ADDR	0
95	BRD_SBUS_OUT	0	**555**	RNGFND1_FUNCTION	0
96	BRD_SD_SLOWDOWN	0	**556**	RNGFND1_GNDCLEAR	10
97	BRD_SER1_RTSCTS	0	**557**	RNGFND1_MAX_CM	700
98	BRD_SER2_RTSCTS	0	**558**	RNGFND1_MIN_CM	20
99	BRD_SERIAL_NUM	0	**559**	RNGFND1_OFFSET	0
100	BRD_TYPE	24	**560**	RNGFND1_ORIENT	0
101	BRD_VBUS_MIN	4.3	**561**	RNGFND1_PIN	-1

102	BRD_VSERVO_MIN	0	**562**	RNGFND1_POS_X	0
103	BTN_ENABLE	0	**563**	RNGFND1_POS_Y	0
104	CAM_AUTO_ONLY	0	**564**	RNGFND1_POS_Z	0
105	CAM_DURATION	10	**565**	RNGFND1_PWRRNG	0
106	CAM_FEEDBACK_PIN	-1	**566**	RNGFND1_RMETRIC	1
107	CAM_FEEDBACK_POL	1	**567**	RNGFND1_SCALING	3
108	CAM_MAX_ROLL	0	**568**	RNGFND1_STOP_PIN	-1
109	CAM_MIN_INTERVAL	0	**569**	RNGFND1_TYPE	0
110	CAM_RELAY_ON	1	**570**	RNGFND2_ADDR	0
111	CAM_SERVO_OFF	1100	**571**	RNGFND2_FUNCTION	0
112	CAM_SERVO_ON	1300	**572**	RNGFND2_GNDCLEAR	10
113	CAM_TRIGG_DIST	0	**573**	RNGFND2_MAX_CM	700
114	CAM_TRIGG_TYPE	0	**574**	RNGFND2_MIN_CM	20
115	CAM_TYPE	0	**575**	RNGFND2_OFFSET	0
116	CAN_D1_PROTOCOL	1	**576**	RNGFND2_ORIENT	0
117	CAN_D2_PROTOCOL	1	**577**	RNGFND2_PIN	-1
118	CAN_P1_DRIVER	0	**578**	RNGFND2_POS_X	0
119	CAN_P2_DRIVER	0	**579**	RNGFND2_POS_Y	0
120	CAN_SLCAN_CPORT	0	**580**	RNGFND2_POS_Z	0
121	CAN_SLCAN_SERNUM	-1	**581**	RNGFND2_PWRRNG	0
122	CAN_SLCAN_TIMOUT	0	**582**	RNGFND2_RMETRIC	1
123	COMPASS_AUTO_ROT	2	**583**	RNGFND2_SCALING	3
124	COMPASS_AUTODEC	1	**584**	RNGFND2_STOP_PIN	-1
125	COMPASS_CAL_FIT	32	**585**	RNGFND2_TYPE	0
126	COMPASS_DEC	-0.2289	**586**	RNGFND3_ADDR	0
127	COMPASS_DEV_ID	658953	**587**	RNGFND3_FUNCTION	0
128	COMPASS_DEV_ID2	658945	**588**	RNGFND3_GNDCLEAR	10
129	COMPASS_DEV_ID3	0	**589**	RNGFND3_MAX_CM	700
130	COMPASS_DIA_X	0.963464	**590**	RNGFND3_MIN_CM	20
131	COMPASS_DIA_Y	0.958935	**591**	RNGFND3_OFFSET	0
132	COMPASS_DIA_Z	1.042703	**592**	RNGFND3_ORIENT	0
133	COMPASS_DIA2_X	1.004654	**593**	RNGFND3_PIN	-1
134	COMPASS_DIA2_Y	1.04455	**594**	RNGFND3_POS_X	0
135	COMPASS_DIA2_Z	1.087546	**595**	RNGFND3_POS_Y	0
136	COMPASS_DIA3_X	0	**596**	RNGFND3_POS_Z	0
137	COMPASS_DIA3_Y	0	**597**	RNGFND3_PWRRNG	0
138	COMPASS_DIA3_Z	0	**598**	RNGFND3_RMETRIC	1
139	COMPASS_ENABLE	1	**599**	RNGFND3_SCALING	3
140	COMPASS_EXP_DID	-1	**600**	RNGFND3_STOP_PIN	-1
141	COMPASS_EXP_DID2	-1	**601**	RNGFND3_TYPE	0
142	COMPASS_EXP_DID3	-1	**602**	RNGFND4_ADDR	0
143	COMPASS_EXTERN2	0	**603**	RNGFND4_FUNCTION	0

144	COMPASS_EXTERN3	0	**604**	RNGFND4_GNDCLEAR	10
145	COMPASS_EXTERNAL	1	**605**	RNGFND4_MAX_CM	700
146	COMPASS_FLTR_RNG	0	**606**	RNGFND4_MIN_CM	20
147	COMPASS_LEARN	0	**607**	RNGFND4_OFFSET	0
148	COMPASS_MOT_X	0	**608**	RNGFND4_ORIENT	0
149	COMPASS_MOT_Y	0	**609**	RNGFND4_PIN	-1
150	COMPASS_MOT_Z	0	**610**	RNGFND4_POS_X	0
151	COMPASS_MOT2_X	0	**611**	RNGFND4_POS_Y	0
152	COMPASS_MOT2_Y	0	**612**	RNGFND4_POS_Z	0
153	COMPASS_MOT2_Z	0	**613**	RNGFND4_PWRRNG	0
154	COMPASS_MOT3_X	0	**614**	RNGFND4_RMETRIC	1
155	COMPASS_MOT3_Y	0	**615**	RNGFND4_SCALING	3
156	COMPASS_MOT3_Z	0	**616**	RNGFND4_STOP_PIN	-1
157	COMPASS_MOTCT	0	**617**	RNGFND4_TYPE	0
158	COMPASS_ODI_X	0.000969	**618**	RNGFND5_ADDR	0
159	COMPASS_ODI_Y	-0.06673	**619**	RNGFND5_FUNCTION	0
160	COMPASS_ODI_Z	0.118935	**620**	RNGFND5_GNDCLEAR	10
161	COMPASS_ODI2_X	-0.01676	**621**	RNGFND5_MAX_CM	700
162	COMPASS_ODI2_Y	-0.09601	**622**	RNGFND5_MIN_CM	20
163	COMPASS_ODI2_Z	0.063797	**623**	RNGFND5_OFFSET	0
164	COMPASS_ODI3_X	0	**624**	RNGFND5_ORIENT	0
165	COMPASS_ODI3_Y	0	**625**	RNGFND5_PIN	-1
166	COMPASS_ODI3_Z	0	**626**	RNGFND5_POS_X	0
167	COMPASS_OFFS_MAX	800	**627**	RNGFND5_POS_Y	0
168	COMPASS_OFS_X	5.793118	**628**	RNGFND5_POS_Z	0
169	COMPASS_OFS_Y	21.75474	**629**	RNGFND5_PWRRNG	0
170	COMPASS_OFS_Z	-102.489	**630**	RNGFND5_RMETRIC	1
171	COMPASS_OFS2_X	28.66513	**631**	RNGFND5_SCALING	3
172	COMPASS_OFS2_Y	98.94672	**632**	RNGFND5_STOP_PIN	-1
173	COMPASS_OFS2_Z	-7.05574	**633**	RNGFND5_TYPE	0
174	COMPASS_OFS3_X	0	**634**	RNGFND6_ADDR	0
175	COMPASS_OFS3_Y	0	**635**	RNGFND6_FUNCTION	0
176	COMPASS_OFS3_Z	0	**636**	RNGFND6_GNDCLEAR	10
177	COMPASS_ORIENT	0	**637**	RNGFND6_MAX_CM	700
178	COMPASS_ORIENT2	0	**638**	RNGFND6_MIN_CM	20
179	COMPASS_ORIENT3	0	**639**	RNGFND6_OFFSET	0
180	COMPASS_PMOT_EN	0	**640**	RNGFND6_ORIENT	0
181	COMPASS_PRIMARY	0	**641**	RNGFND6_PIN	-1
182	COMPASS_TYPEMASK	0	**642**	RNGFND6_POS_X	0
183	COMPASS_USE	1	**643**	RNGFND6_POS_Y	0
184	COMPASS_USE2	0	**644**	RNGFND6_POS_Z	0
185	COMPASS_USE3	0	**645**	RNGFND6_PWRRNG	0

186	CRASH_ANGLE	0	**646**	RNGFND6_RMETRIC	1
187	CRUISE_SPEED	1.111076	**647**	RNGFND6_SCALING	3
188	CRUISE_THROTTLE	100	**648**	RNGFND6_STOP_PIN	-1
189	EK2_ABIAS_P_NSE	0.005	**649**	RNGFND6_TYPE	0
190	EK2_ACC_P_NSE	0.6	**650**	RNGFND7_ADDR	0
191	EK2_ALT_M_NSE	3	**651**	RNGFND7_FUNCTION	0
192	EK2_ALT_SOURCE	0	**652**	RNGFND7_GNDCLEAR	10
193	EK2_BCN_DELAY	50	**653**	RNGFND7_MAX_CM	700
194	EK2_BCN_I_GTE	500	**654**	RNGFND7_MIN_CM	20
195	EK2_BCN_M_NSE	1	**655**	RNGFND7_OFFSET	0
196	EK2_CHECK_SCALE	100	**656**	RNGFND7_ORIENT	0
197	EK2_EAS_I_GATE	400	**657**	RNGFND7_PIN	-1
198	EK2_EAS_M_NSE	1.4	**658**	RNGFND7_POS_X	0
199	EK2_ENABLE	1	**659**	RNGFND7_POS_Y	0
200	EK2_EXTNAV_DELAY	10	**660**	RNGFND7_POS_Z	0
201	EK2_FLOW_DELAY	10	**661**	RNGFND7_PWRRNG	0
202	EK2_FLOW_I_GATE	300	**662**	RNGFND7_RMETRIC	1
203	EK2_FLOW_M_NSE	0.25	**663**	RNGFND7_SCALING	3
204	EK2_FLOW_USE	1	**664**	RNGFND7_STOP_PIN	-1
205	EK2_GBIAS_P_NSE	0.0001	**665**	RNGFND7_TYPE	0
206	EK2_GLITCH_RAD	25	**666**	RNGFND8_ADDR	0
207	EK2_GPS_CHECK	31	**667**	RNGFND8_FUNCTION	0
208	EK2_GPS_TYPE	1	**668**	RNGFND8_GNDCLEAR	10
209	EK2_GSCL_P_NSE	0.0005	**669**	RNGFND8_MAX_CM	700
210	EK2_GYRO_P_NSE	0.03	**670**	RNGFND8_MIN_CM	20
211	EK2_HGT_DELAY	60	**671**	RNGFND8_OFFSET	0
212	EK2_HGT_I_GATE	500	**672**	RNGFND8_ORIENT	0
213	EK2_HRT_FILT	2	**673**	RNGFND8_PIN	-1
214	EK2_IMU_MASK	3	**674**	RNGFND8_POS_X	0
215	EK2_LOG_MASK	1	**675**	RNGFND8_POS_Y	0
216	EK2_MAG_CAL	2	**676**	RNGFND8_POS_Z	0
217	EK2_MAG_EF_LIM	50	**677**	RNGFND8_PWRRNG	0
218	EK2_MAG_I_GATE	300	**678**	RNGFND8_RMETRIC	1
219	EK2_MAG_M_NSE	0.05	**679**	RNGFND8_SCALING	3
220	EK2_MAG_MASK	0	**680**	RNGFND8_STOP_PIN	-1
221	EK2_MAGB_P_NSE	0.0001	**681**	RNGFND8_TYPE	0
222	EK2_MAGE_P_NSE	0.001	**682**	RNGFND9_ADDR	0
223	EK2_MAX_FLOW	2.5	**683**	RNGFND9_FUNCTION	0
224	EK2_NOAID_M_NSE	10	**684**	RNGFND9_GNDCLEAR	10
225	EK2_OGN_HGT_MASK	0	**685**	RNGFND9_MAX_CM	700
226	EK2_POS_I_GATE	500	**686**	RNGFND9_MIN_CM	20
227	EK2_POSNE_M_NSE	1	**687**	RNGFND9_OFFSET	0

228	EK2_RNG_I_GATE	500	**688**	RNGFND9_ORIENT	0
229	EK2_RNG_M_NSE	0.5	**689**	RNGFND9_PIN	-1
230	EK2_RNG_USE_HGT	-1	**690**	RNGFND9_POS_X	0
231	EK2_RNG_USE_SPD	2	**691**	RNGFND9_POS_Y	0
232	EK2_TAU_OUTPUT	25	**692**	RNGFND9_POS_Z	0
233	EK2_TERR_GRAD	0.1	**693**	RNGFND9_PWRRNG	0
234	EK2_VEL_I_GATE	500	**694**	RNGFND9_RMETRIC	1
235	EK2_VELD_M_NSE	0.7	**695**	RNGFND9_SCALING	3
236	EK2_VELNE_M_NSE	0.5	**696**	RNGFND9_STOP_PIN	-1
237	EK2_WIND_P_NSE	0.1	**697**	RNGFND9_TYPE	0
238	EK2_WIND_PSCALE	0.5	**698**	RNGFNDA_ADDR	0
239	EK2_YAW_I_GATE	300	**699**	RNGFNDA_FUNCTION	0
240	EK2_YAW_M_NSE	0.5	**700**	RNGFNDA_GNDCLEAR	10
241	EK3_ENABLE	0	**701**	RNGFNDA_MAX_CM	700
242	FENCE_ACTION	1	**702**	RNGFNDA_MIN_CM	20
243	FENCE_ENABLE	0	**703**	RNGFNDA_OFFSET	0
244	FENCE_MARGIN	2	**704**	RNGFNDA_ORIENT	0
245	FENCE_RADIUS	300	**705**	RNGFNDA_PIN	-1
246	FENCE_TOTAL	0	**706**	RNGFNDA_POS_X	0
247	FENCE_TYPE	6	**707**	RNGFNDA_POS_Y	0
248	FOLL_ENABLE	0	**708**	RNGFNDA_POS_Z	0
249	FORMAT_VERSION	16	**709**	RNGFNDA_PWRRNG	0
250	FRAME_CLASS	1	**710**	RNGFNDA_RMETRIC	1
251	FRAME_TYPE	0	**711**	RNGFNDA_SCALING	3
252	FS_ACTION	2	**712**	RNGFNDA_STOP_PIN	-1
253	FS_CRASH_CHECK	0	**713**	RNGFNDA_TYPE	0
254	FS_EKF_ACTION	1	**714**	RPM_MAX	100000
255	FS_EKF_THRESH	0.8	**715**	RPM_MIN	10
256	FS_GCS_ENABLE	0	**716**	RPM_MIN_QUAL	0.5
257	FS_OPTIONS	0	**717**	RPM_PIN	54
258	FS_THR_ENABLE	1	**718**	RPM_SCALING	1
259	FS_THR_VALUE	910	**719**	RPM_TYPE	0
260	FS_TIMEOUT	1.5	**720**	RPM2_PIN	-1
261	GCS_PID_MASK	0	**721**	RPM2_SCALING	1
262	GND_ABS_PRESS	97793.36	**722**	RPM2_TYPE	0
263	GND_ABS_PRESS2	0	**723**	RSSI_TYPE	0
264	GND_ABS_PRESS3	0	**724**	RST_SWITCH_CH	0
265	GND_ALT_OFFSET	0	**725**	RTL_SPEED	0
266	GND_EXT_BUS	-1	**726**	SAIL_ENABLE	0
267	GND_FLTR_RNG	0	**727**	SCHED_DEBUG	0
268	GND_PRIMARY	0	**728**	SCHED_LOOP_RATE	50
269	GND_PROBE_EXT	0	**729**	SCR_ENABLE	0

270	GND_TEMP	0	**730**	SERIAL_PASS1	0
271	GPS_AUTO_CONFIG	1	**731**	SERIAL_PASS2	-1
272	GPS_AUTO_SWITCH	1	**732**	SERIAL_PASSTIMO	15
273	GPS_BLEND_MASK	5	**733**	SERIAL0_BAUD	115
274	GPS_BLEND_TC	10	**734**	SERIAL0_PROTOCOL	2
275	GPS_DELAY_MS	0	**735**	SERIAL1_BAUD	57
276	GPS_DELAY_MS2	0	**736**	SERIAL1_OPTIONS	0
277	GPS_GNSS_MODE	0	**737**	SERIAL1_PROTOCOL	1
278	GPS_GNSS_MODE2	0	**738**	SERIAL2_BAUD	57
279	GPS_INJECT_TO	127	**739**	SERIAL2_OPTIONS	0
280	GPS_MIN_DGPS	100	**740**	SERIAL2_PROTOCOL	1
281	GPS_MIN_ELEV	-100	**741**	SERIAL3_BAUD	38
282	GPS_NAVFILTER	8	**742**	SERIAL3_OPTIONS	0
283	GPS_POS1_X	0.185	**743**	SERIAL3_PROTOCOL	5
284	GPS_POS1_Y	0	**744**	SERIAL4_BAUD	38
285	GPS_POS1_Z	-0.175	**745**	SERIAL4_OPTIONS	0
286	GPS_POS2_X	-0.125	**746**	SERIAL4_PROTOCOL	5
287	GPS_POS2_Y	0	**747**	SERIAL5_BAUD	57
288	GPS_POS2_Z	-0.11	**748**	SERIAL5_OPTIONS	0
289	GPS_RATE_MS	200	**749**	SERIAL5_PROTOCOL	-1
290	GPS_RATE_MS2	200	**750**	SERIAL6_BAUD	57
291	GPS_RAW_DATA	0	**751**	SERIAL6_OPTIONS	0
292	GPS_SAVE_CFG	2	**752**	SERIAL6_PROTOCOL	-1
293	GPS_SBAS_MODE	2	**753**	SERIAL7_BAUD	115200
294	GPS_SBP_LOGMASK	-256	**754**	SERIAL7_OPTIONS	0
295	GPS_TYPE	1	**755**	SERIAL7_PROTOCOL	2
296	GPS_TYPE2	0	**756**	SERVO_BLH_DEBUG	0
297	GRIP_ENABLE	0	**757**	SERVO_BLH_MASK	0
298	INITIAL_MODE	0	**758**	SERVO_BLH_OTYPE	0
299	INS_ACC_BODYFIX	2	**759**	SERVO_BLH_POLES	14
300	INS_ACC_ID	2621706	**760**	SERVO_BLH_PORT	0
301	INS_ACC2_ID	2688010	**761**	SERVO_BLH_REMASK	0
302	INS_ACC2OFFS_X	0.161591	**762**	SERVO_BLH_TEST	0
303	INS_ACC2OFFS_Y	-0.03968	**763**	SERVO_BLH_TMOUT	0
304	INS_ACC2OFFS_Z	0.138354	**764**	SERVO_BLH_TRATE	10
305	INS_ACC2SCAL_X	0.991839	**765**	SERVO_RATE	50
306	INS_ACC2SCAL_Y	0.990431	**766**	SERVO_ROB_POSMAX	4095
307	INS_ACC2SCAL_Z	0.981222	**767**	SERVO_ROB_POSMIN	0
308	INS_ACC3_ID	0	**768**	SERVO_SBUS_RATE	50
309	INS_ACC3OFFS_X	0	**769**	SERVO_VOLZ_MASK	0
310	INS_ACC3OFFS_Y	0	**770**	SERVO1_FUNCTION	73
311	INS_ACC3OFFS_Z	0	**771**	SERVO1_MAX	1880

312	INS_ACC3SCAL_X	0	**772**	SERVO1_MIN	1100
313	INS_ACC3SCAL_Y	0	**773**	SERVO1_REVERSED	0
314	INS_ACC3SCAL_Z	0	**774**	SERVO1_TRIM	1500
315	INS_ACCEL_FILTER	10	**775**	SERVO10_FUNCTION	0
316	INS_ACCOFFS_X	-0.08955	**776**	SERVO10_MAX	1900
317	INS_ACCOFFS_Y	0.172145	**777**	SERVO10_MIN	1100
318	INS_ACCOFFS_Z	0.064103	**778**	SERVO10_REVERSED	0
319	INS_ACCSCAL_X	0.997815	**779**	SERVO10_TRIM	1500
320	INS_ACCSCAL_Y	0.999581	**780**	SERVO11_FUNCTION	0
321	INS_ACCSCAL_Z	0.98419	**781**	SERVO11_MAX	1900
322	INS_ENABLE_MASK	127	**782**	SERVO11_MIN	1100
323	INS_FAST_SAMPLE	1	**783**	SERVO11_REVERSED	0
324	INS_GYR_CAL	1	**784**	SERVO11_TRIM	1500
325	INS_GYR_ID	2621706	**785**	SERVO12_FUNCTION	0
326	INS_GYR2_ID	2687754	**786**	SERVO12_MAX	1900
327	INS_GYR2OFFS_X	-0.00299	**787**	SERVO12_MIN	1100
328	INS_GYR2OFFS_Y	0.003449	**788**	SERVO12_REVERSED	0
329	INS_GYR2OFFS_Z	0.001118	**789**	SERVO12_TRIM	1500
330	INS_GYR3_ID	0	**790**	SERVO13_FUNCTION	0
331	INS_GYR3OFFS_X	0	**791**	SERVO13_MAX	1900
332	INS_GYR3OFFS_Y	0	**792**	SERVO13_MIN	1100
333	INS_GYR3OFFS_Z	0	**793**	SERVO13_REVERSED	0
334	INS_GYRO_FILTER	4	**794**	SERVO13_TRIM	1500
335	INS_GYROFFS_X	0.025607	**795**	SERVO14_FUNCTION	0
336	INS_GYROFFS_Y	-0.01907	**796**	SERVO14_MAX	1900
337	INS_GYROFFS_Z	-0.00255	**797**	SERVO14_MIN	1100
338	INS_HNTCH_ENABLE	0	**798**	SERVO14_REVERSED	0
339	INS_LOG_BAT_CNT	1024	**799**	SERVO14_TRIM	1500
340	INS_LOG_BAT_LGCT	32	**800**	SERVO15_FUNCTION	0
341	INS_LOG_BAT_LGIN	20	**801**	SERVO15_MAX	1900
342	INS_LOG_BAT_MASK	0	**802**	SERVO15_MIN	1100
343	INS_LOG_BAT_OPT	0	**803**	SERVO15_REVERSED	0
344	INS_NOTCH_ENABLE	0	**804**	SERVO15_TRIM	1500
345	INS_POS1_X	0.04	**805**	SERVO16_FUNCTION	0
346	INS_POS1_Y	0.05	**806**	SERVO16_MAX	1900
347	INS_POS1_Z	0.045	**807**	SERVO16_MIN	1100
348	INS_POS2_X	0	**808**	SERVO16_REVERSED	0
349	INS_POS2_Y	0	**809**	SERVO16_TRIM	1500
350	INS_POS2_Z	0	**810**	SERVO2_FUNCTION	0
351	INS_POS3_X	0	**811**	SERVO2_MAX	1900
352	INS_POS3_Y	0	**812**	SERVO2_MIN	1100
353	INS_POS3_Z	0	**813**	SERVO2_REVERSED	0

354	INS_STILL_THRESH	0.1	**814**	SERVO2_TRIM	1500
355	INS_TRIM_OPTION	1	**815**	SERVO3_FUNCTION	74
356	INS_USE	1	**816**	SERVO3_MAX	1950
357	INS_USE2	1	**817**	SERVO3_MIN	1100
358	INS_USE3	1	**818**	SERVO3_REVERSED	0
359	LOG_BACKEND_TYPE	1	**819**	SERVO3_TRIM	1500
360	LOG_BITMASK	65535	**820**	SERVO4_FUNCTION	0
361	LOG_DISARMED	1	**821**	SERVO4_MAX	1900
362	LOG_FILE_BUFSIZE	50	**822**	SERVO4_MIN	1100
363	LOG_FILE_DSRMROT	0	**823**	SERVO4_REVERSED	0
364	LOG_FILE_TIMEOUT	5	**824**	SERVO4_TRIM	1500
365	LOG_MAV_BUFSIZE	8	**825**	SERVO5_FUNCTION	0
366	LOG_REPLAY	0	**826**	SERVO5_MAX	1900
367	LOIT_RADIUS	2	**827**	SERVO5_MIN	1100
368	LOIT_SPEED_GAIN	0.5	**828**	SERVO5_REVERSED	0
369	LOIT_TYPE	0	**829**	SERVO5_TRIM	1500
370	MIS_DONE_BEHAVE	0	**830**	SERVO6_FUNCTION	0
371	MIS_OPTIONS	0	**831**	SERVO6_MAX	1900
372	MIS_RESTART	0	**832**	SERVO6_MIN	1100
373	MIS_TOTAL	36	**833**	SERVO6_REVERSED	0
374	MNT_ANGMAX_PAN	4500	**834**	SERVO6_TRIM	1500
375	MNT_ANGMAX_ROL	4500	**835**	SERVO7_FUNCTION	0
376	MNT_ANGMAX_TIL	4500	**836**	SERVO7_MAX	1900
377	MNT_ANGMIN_PAN	-4500	**837**	SERVO7_MIN	1100
378	MNT_ANGMIN_ROL	-4500	**838**	SERVO7_REVERSED	0
379	MNT_ANGMIN_TIL	-4500	**839**	SERVO7_TRIM	1500
380	MNT_DEFLT_MODE	3	**840**	SERVO8_FUNCTION	0
381	MNT_JSTICK_SPD	0	**841**	SERVO8_MAX	1900
382	MNT_LEAD_PTCH	0	**842**	SERVO8_MIN	1100
383	MNT_LEAD_RLL	0	**843**	SERVO8_REVERSED	0
384	MNT_NEUTRAL_X	0	**844**	SERVO8_TRIM	1500
385	MNT_NEUTRAL_Y	0	**845**	SERVO9_FUNCTION	0
386	MNT_NEUTRAL_Z	0	**846**	SERVO9_MAX	1900
387	MNT_RC_IN_PAN	0	**847**	SERVO9_MIN	1100
388	MNT_RC_IN_ROLL	0	**848**	SERVO9_REVERSED	0
389	MNT_RC_IN_TILT	0	**849**	SERVO9_TRIM	1500
390	MNT_RETRACT_X	0	**850**	SIMPLE_TYPE	0
391	MNT_RETRACT_Y	0	**851**	SPEED_MAX	0
392	MNT_RETRACT_Z	0	**852**	SPRAY_ENABLE	0
393	MNT_STAB_PAN	0	**853**	SR0_ADSB	0
394	MNT_STAB_ROLL	0	**854**	SR0_EXT_STAT	2
395	MNT_STAB_TILT	0	**855**	SR0_EXTRA1	4

396	MNT_TYPE	0	**856**	SR0_EXTRA2	4
397	MODE_CH	6	**857**	SR0_EXTRA3	2
398	MODE1	10	**858**	SR0_PARAMS	10
399	MODE2	4	**859**	SR0_POSITION	2
400	MODE3	5	**860**	SR0_RAW_CTRL	1
401	MODE4	0	**861**	SR0_RAW_SENS	2
402	MODE5	0	**862**	SR0_RC_CHAN	2
403	MODE6	3	**863**	SR1_ADSB	0
404	MOT_PWM_FREQ	16	**864**	SR1_EXT_STAT	2
405	MOT_PWM_TYPE	0	**865**	SR1_EXTRA1	4
406	MOT_SAFE_DISARM	0	**866**	SR1_EXTRA2	4
407	MOT_SLEWRATE	100	**867**	SR1_EXTRA3	2
408	MOT_SPD_SCA_BASE	1	**868**	SR1_PARAMS	10
409	MOT_THR_MAX	100	**869**	SR1_POSITION	2
410	MOT_THR_MIN	4	**870**	SR1_RAW_CTRL	1
411	MOT_THST_EXPO	0	**871**	SR1_RAW_SENS	2
412	MOT_VEC_THR_BASE	0	**872**	SR1_RC_CHAN	2
413	NAVL1_DAMPING	0.75	**873**	SR2_ADSB	0
414	NAVL1_PERIOD	11	**874**	SR2_EXT_STAT	1
415	NAVL1_XTRACK_I	0.02	**875**	SR2_EXTRA1	1
416	NTF_BUZZ_ENABLE	1	**876**	SR2_EXTRA2	1
417	NTF_BUZZ_ON_LVL	1	**877**	SR2_EXTRA3	1
418	NTF_BUZZ_PIN	0	**878**	SR2_PARAMS	10
419	NTF_BUZZ_VOLUME	100	**879**	SR2_POSITION	1
420	NTF_DISPLAY_TYPE	0	**880**	SR2_RAW_CTRL	1
421	NTF_LED_BRIGHT	3	**881**	SR2_RAW_SENS	1
422	NTF_LED_OVERRIDE	0	**882**	SR2_RC_CHAN	1
423	NTF_LED_TYPES	199	**883**	SR3_ADSB	0
424	NTF_OREO_THEME	0	**884**	SR3_EXT_STAT	2
425	OA_TYPE	0	**885**	SR3_EXTRA1	4
426	PILOT_STEER_TYPE	0	**886**	SR3_EXTRA2	4
427	PRX_IGN_ANG1	0	**887**	SR3_EXTRA3	2
428	PRX_IGN_ANG2	0	**888**	SR3_PARAMS	10
429	PRX_IGN_ANG3	0	**889**	SR3_POSITION	2
430	PRX_IGN_ANG4	0	**890**	SR3_RAW_CTRL	1
431	PRX_IGN_ANG5	0	**891**	SR3_RAW_SENS	2
432	PRX_IGN_ANG6	0	**892**	SR3_RC_CHAN	2
433	PRX_IGN_WID1	0	**893**	SRTL_ACCURACY	2
434	PRX_IGN_WID2	0	**894**	SRTL_POINTS	300
435	PRX_IGN_WID3	0	**895**	STAT_BOOTCNT	17
436	PRX_IGN_WID4	0	**896**	STAT_FLTTIME	2257
437	PRX_IGN_WID5	0	**897**	STAT_RESET	1.6E+08

438	PRX_IGN_WID6	0	**898**	STAT_RUNTIME	185820
439	PRX_ORIENT	0	**899**	STICK_MIXING	0
440	PRX_TYPE	0	**900**	SYSID_ENFORCE	0
441	PRX_YAW_CORR	0	**901**	SYSID_MYGCS	255
442	RALLY_INCL_HOME	1	**902**	SYSID_THISMAV	1
443	RALLY_LIMIT_KM	0.5	**903**	TELEM_DELAY	0
444	RALLY_TOTAL	0	**904**	TURN_MAX_G	0.6
445	RC_OPTIONS	0	**905**	TURN_RADIUS	0.1
446	RC_OVERRIDE_TIME	3	**906**	VISO_ORIENT	0
447	RC1_DZ	0	**907**	VISO_POS_X	0
448	RC1_MAX	1901	**908**	VISO_POS_Y	0
449	RC1_MIN	1099	**909**	VISO_POS_Z	0
450	RC1_OPTION	0	**910**	VISO_TYPE	0
451	RC1_REVERSED	0	**911**	WENC_TYPE	0
452	RC1_TRIM	1099	**912**	WNDVN_TYPE	0
453	RC10_DZ	0	**913**	WP_OVERSHOOT	2
454	RC10_MAX	2000	**914**	WP_PIVOT_ANGLE	60
455	RC10_MIN	1000	**915**	WP_PIVOT_RATE	90
456	RC10_OPTION	0	**916**	WP_RADIUS	2
457	RC10_REVERSED	0	**917**	WP_SPEED	1
458	RC10_TRIM	1500	**918**	WP_SPEED_MIN	0
459	RC11_DZ	0	**919**	WRC_ENABLE	0
460	RC11_MAX	2000			

A.2 RTK GPS Configuration

In this configuration, RTK GPS is used with the Pixhawk. The procedure for saving this parameter list into a format compatible with Mission Planner is described in Appendix A.1.

Index	*Parameter*	*Value*	*Index*	*Parameter*	*Value*
1	ACRO_TURN_RATE	180	**461**	RC11_MIN	1000
2	AHRS_COMP_BETA	0.1	**462**	RC11_OPTION	0
3	AHRS_CUSTOM_PIT	0	**463**	RC11_REVERSED	0
4	AHRS_CUSTOM_ROLL	0	**464**	RC11_TRIM	1500
5	AHRS_CUSTOM_YAW	0	**465**	RC12_DZ	0
6	AHRS_EKF_TYPE	2	**466**	RC12_MAX	2000
7	AHRS_GPS_GAIN	1	**467**	RC12_MIN	1000
8	AHRS_GPS_MINSATS	6	**468**	RC12_OPTION	0
9	AHRS_GPS_USE	1	**469**	RC12_REVERSED	0
10	AHRS_ORIENTATION	0	**470**	RC12_TRIM	1500
11	AHRS_RP_P	0.2	**471**	RC13_DZ	0
12	AHRS_TRIM_X	-0.01955	**472**	RC13_MAX	2000
13	AHRS_TRIM_Y	0.027536	**473**	RC13_MIN	1000
14	AHRS_TRIM_Z	0	**474**	RC13_OPTION	0
15	AHRS_WIND_MAX	0	**475**	RC13_REVERSED	0
16	AHRS_YAW_P	0.2	**476**	RC13_TRIM	1500
17	ARMING_ACCTHRESH	0.75	**477**	RC14_DZ	0
18	ARMING_CHECK	60926	**478**	RC14_MAX	2000
19	ARMING_MIS_ITEMS	0	**479**	RC14_MIN	1000
20	ARMING_REQUIRE	1	**480**	RC14_OPTION	0
21	ARMING_RUDDER	2	**481**	RC14_REVERSED	0
22	ARSPD_TYPE	0	**482**	RC14_TRIM	1500
23	ATC_ACCEL_MAX	0.3	**483**	RC15_DZ	0
24	ATC_BAL_D	0.03	**484**	RC15_MAX	1900
25	ATC_BAL_FF	0	**485**	RC15_MIN	1100
26	ATC_BAL_FLTD	0	**486**	RC15_OPTION	0
27	ATC_BAL_FLTE	10	**487**	RC15_REVERSED	0
28	ATC_BAL_FLTT	0	**488**	RC15_TRIM	1500
29	ATC_BAL_I	1.5	**489**	RC16_DZ	0
30	ATC_BAL_IMAX	1	**490**	RC16_MAX	1900
31	ATC_BAL_P	1.8	**491**	RC16_MIN	1100
32	ATC_BAL_SPD_FF	1	**492**	RC16_OPTION	0
33	ATC_BRAKE	0	**493**	RC16_REVERSED	0
34	ATC_DECEL_MAX	0	**494**	RC16_TRIM	1500
35	ATC_SAIL_D	0	**495**	RC2_DZ	30

36	ATC_SAIL_FF	0	**496**	RC2_MAX	1898
37	ATC_SAIL_FLTD	0	**497**	RC2_MIN	1098
38	ATC_SAIL_FLTE	10	**498**	RC2_OPTION	0
39	ATC_SAIL_FLTT	0	**499**	RC2_REVERSED	0
40	ATC_SAIL_I	0.1	**500**	RC2_TRIM	1498
41	ATC_SAIL_IMAX	1	**501**	RC3_DZ	0
42	ATC_SAIL_P	1	**502**	RC3_MAX	1900
43	ATC_SPEED_D	0	**503**	RC3_MIN	1100
44	ATC_SPEED_FF	0	**504**	RC3_OPTION	0
45	ATC_SPEED_FLTD	0	**505**	RC3_REVERSED	0
46	ATC_SPEED_FLTE	10	**506**	RC3_TRIM	1500
47	ATC_SPEED_FLTT	0	**507**	RC4_DZ	30
48	ATC_SPEED_I	0.2	**508**	RC4_MAX	1902
49	ATC_SPEED_IMAX	1	**509**	RC4_MIN	1102
50	ATC_SPEED_P	0.2	**510**	RC4_OPTION	0
51	ATC_STOP_SPEED	0.1	**511**	RC4_REVERSED	1
52	ATC_STR_ACC_MAX	180	**512**	RC4_TRIM	1502
53	ATC_STR_ANG_P	2.5	**513**	RC5_DZ	0
54	ATC_STR_RAT_D	0	**514**	RC5_MAX	1901
55	ATC_STR_RAT_FF	0.2	**515**	RC5_MIN	1099
56	ATC_STR_RAT_FLTD	0	**516**	RC5_OPTION	50
57	ATC_STR_RAT_FLTE	10	**517**	RC5_REVERSED	0
58	ATC_STR_RAT_FLTT	0	**518**	RC5_TRIM	1500
59	ATC_STR_RAT_I	0.2	**519**	RC6_DZ	0
60	ATC_STR_RAT_IMAX	0.8	**520**	RC6_MAX	1901
61	ATC_STR_RAT_MAX	360	**521**	RC6_MIN	1099
62	ATC_STR_RAT_P	0.1	**522**	RC6_OPTION	0
63	AUTO_KICKSTART	0	**523**	RC6_REVERSED	0
64	AUTO_TRIGGER_PIN	-1	**524**	RC6_TRIM	1500
65	AVOID_ANGLE_MAX	1000	**525**	RC7_DZ	0
66	AVOID_BEHAVE	1	**526**	RC7_MAX	1901
67	AVOID_DIST_MAX	5	**527**	RC7_MIN	1099
68	AVOID_ENABLE	3	**528**	RC7_OPTION	0
69	AVOID_MARGIN	2	**529**	RC7_REVERSED	0
70	BAL_PITCH_MAX	2	**530**	RC7_TRIM	1500
71	BAL_PITCH_TRIM	0	**531**	RC8_DZ	0
72	BATT_MONITOR	0	**532**	RC8_MAX	1901
73	BATT2_MONITOR	0	**533**	RC8_MIN	1099
74	BATT3_MONITOR	0	**534**	RC8_OPTION	41
75	BATT4_MONITOR	0	**535**	RC8_REVERSED	0
76	BATT5_MONITOR	0	**536**	RC8_TRIM	1500
77	BATT6_MONITOR	0	**537**	RC9_DZ	0

78	BATT7_MONITOR	0	**538**	RC9_MAX	2000
79	BATT8_MONITOR	0	**539**	RC9_MIN	1000
80	BATT9_MONITOR	0	**540**	RC9_OPTION	0
81	BCN_ALT	0	**541**	RC9_REVERSED	0
82	BCN_LATITUDE	0	**542**	RC9_TRIM	1500
83	BCN_LONGITUDE	0	**543**	RCMAP_PITCH	1
84	BCN_ORIENT_YAW	0	**544**	RCMAP_ROLL	2
85	BCN_TYPE	0	**545**	RCMAP_THROTTLE	3
86	BRD_BOOT_DELAY	0	**546**	RCMAP_YAW	4
87	BRD_IO_ENABLE	1	**547**	RELAY_DEFAULT	0
88	BRD_OPTIONS	1	**548**	RELAY_PIN	-1
89	BRD_PWM_COUNT	8	**549**	RELAY_PIN2	-1
90	BRD_RTC_TYPES	1	**550**	RELAY_PIN3	-1
91	BRD_RTC_TZ_MIN	0	**551**	RELAY_PIN4	-1
92	BRD_SAFETY_MASK	0	**552**	RELAY_PIN5	-1
93	BRD_SAFETYENABLE	1	**553**	RELAY_PIN6	-1
94	BRD_SAFETYOPTION	7	**554**	RNGFND1_ADDR	0
95	BRD_SBUS_OUT	0	**555**	RNGFND1_FUNCTION	0
96	BRD_SD_SLOWDOWN	0	**556**	RNGFND1_GNDCLEAR	10
97	BRD_SER1_RTSCTS	0	**557**	RNGFND1_MAX_CM	700
98	BRD_SER2_RTSCTS	0	**558**	RNGFND1_MIN_CM	20
99	BRD_SERIAL_NUM	0	**559**	RNGFND1_OFFSET	0
100	BRD_TYPE	24	**560**	RNGFND1_ORIENT	0
101	BRD_VBUS_MIN	4.3	**561**	RNGFND1_PIN	-1
102	BRD_VSERVO_MIN	0	**562**	RNGFND1_POS_X	0
103	BTN_ENABLE	0	**563**	RNGFND1_POS_Y	0
104	CAM_AUTO_ONLY	0	**564**	RNGFND1_POS_Z	0
105	CAM_DURATION	10	**565**	RNGFND1_PWRRNG	0
106	CAM_FEEDBACK_PIN	-1	**566**	RNGFND1_RMETRIC	1
107	CAM_FEEDBACK_POL	1	**567**	RNGFND1_SCALING	3
108	CAM_MAX_ROLL	0	**568**	RNGFND1_STOP_PIN	-1
109	CAM_MIN_INTERVAL	0	**569**	RNGFND1_TYPE	0
110	CAM_RELAY_ON	1	**570**	RNGFND2_ADDR	0
111	CAM_SERVO_OFF	1100	**571**	RNGFND2_FUNCTION	0
112	CAM_SERVO_ON	1300	**572**	RNGFND2_GNDCLEAR	10
113	CAM_TRIGG_DIST	0	**573**	RNGFND2_MAX_CM	700
114	CAM_TRIGG_TYPE	0	**574**	RNGFND2_MIN_CM	20
115	CAM_TYPE	0	**575**	RNGFND2_OFFSET	0
116	CAN_D1_PROTOCOL	1	**576**	RNGFND2_ORIENT	0
117	CAN_D2_PROTOCOL	1	**577**	RNGFND2_PIN	-1
118	CAN_P1_DRIVER	0	**578**	RNGFND2_POS_X	0
119	CAN_P2_DRIVER	0	**579**	RNGFND2_POS_Y	0

120	CAN_SLCAN_CPORT	0	**580**	RNGFND2_POS_Z	0
121	CAN_SLCAN_SERNUM	-1	**581**	RNGFND2_PWRRNG	0
122	CAN_SLCAN_TIMOUT	0	**582**	RNGFND2_RMETRIC	1
123	COMPASS_AUTO_ROT	2	**583**	RNGFND2_SCALING	3
124	COMPASS_AUTODEC	1	**584**	RNGFND2_STOP_PIN	-1
125	COMPASS_CAL_FIT	32	**585**	RNGFND2_TYPE	0
126	COMPASS_DEC	-0.2289	**586**	RNGFND3_ADDR	0
127	COMPASS_DEV_ID	658953	**587**	RNGFND3_FUNCTION	0
128	COMPASS_DEV_ID2	658945	**588**	RNGFND3_GNDCLEAR	10
129	COMPASS_DEV_ID3	0	**589**	RNGFND3_MAX_CM	700
130	COMPASS_DIA_X	0.963464	**590**	RNGFND3_MIN_CM	20
131	COMPASS_DIA_Y	0.958935	**591**	RNGFND3_OFFSET	0
132	COMPASS_DIA_Z	1.042703	**592**	RNGFND3_ORIENT	0
133	COMPASS_DIA2_X	1.004654	**593**	RNGFND3_PIN	-1
134	COMPASS_DIA2_Y	1.04455	**594**	RNGFND3_POS_X	0
135	COMPASS_DIA2_Z	1.087546	**595**	RNGFND3_POS_Y	0
136	COMPASS_DIA3_X	0	**596**	RNGFND3_POS_Z	0
137	COMPASS_DIA3_Y	0	**597**	RNGFND3_PWRRNG	0
138	COMPASS_DIA3_Z	0	**598**	RNGFND3_RMETRIC	1
139	COMPASS_ENABLE	1	**599**	RNGFND3_SCALING	3
140	COMPASS_EXP_DID	-1	**600**	RNGFND3_STOP_PIN	-1
141	COMPASS_EXP_DID2	-1	**601**	RNGFND3_TYPE	0
142	COMPASS_EXP_DID3	-1	**602**	RNGFND4_ADDR	0
143	COMPASS_EXTERN2	0	**603**	RNGFND4_FUNCTION	0
144	COMPASS_EXTERN3	0	**604**	RNGFND4_GNDCLEAR	10
145	COMPASS_EXTERNAL	1	**605**	RNGFND4_MAX_CM	700
146	COMPASS_FLTR_RNG	0	**606**	RNGFND4_MIN_CM	20
147	COMPASS_LEARN	0	**607**	RNGFND4_OFFSET	0
148	COMPASS_MOT_X	0	**608**	RNGFND4_ORIENT	0
149	COMPASS_MOT_Y	0	**609**	RNGFND4_PIN	-1
150	COMPASS_MOT_Z	0	**610**	RNGFND4_POS_X	0
151	COMPASS_MOT2_X	0	**611**	RNGFND4_POS_Y	0
152	COMPASS_MOT2_Y	0	**612**	RNGFND4_POS_Z	0
153	COMPASS_MOT2_Z	0	**613**	RNGFND4_PWRRNG	0
154	COMPASS_MOT3_X	0	**614**	RNGFND4_RMETRIC	1
155	COMPASS_MOT3_Y	0	**615**	RNGFND4_SCALING	3
156	COMPASS_MOT3_Z	0	**616**	RNGFND4_STOP_PIN	-1
157	COMPASS_MOTCT	0	**617**	RNGFND4_TYPE	0
158	COMPASS_ODI_X	0.000969	**618**	RNGFND5_ADDR	0
159	COMPASS_ODI_Y	-0.06673	**619**	RNGFND5_FUNCTION	0
160	COMPASS_ODI_Z	0.118935	**620**	RNGFND5_GNDCLEAR	10
161	COMPASS_ODI2_X	-0.01676	**621**	RNGFND5_MAX_CM	700

162	COMPASS_ODI2_Y	-0.09601	**622**	RNGFND5_MIN_CM	20
163	COMPASS_ODI2_Z	0.063797	**623**	RNGFND5_OFFSET	0
164	COMPASS_ODI3_X	0	**624**	RNGFND5_ORIENT	0
165	COMPASS_ODI3_Y	0	**625**	RNGFND5_PIN	-1
166	COMPASS_ODI3_Z	0	**626**	RNGFND5_POS_X	0
167	COMPASS_OFFS_MAX	800	**627**	RNGFND5_POS_Y	0
168	COMPASS_OFS_X	5.793118	**628**	RNGFND5_POS_Z	0
169	COMPASS_OFS_Y	21.75474	**629**	RNGFND5_PWRRNG	0
170	COMPASS_OFS_Z	-102.489	**630**	RNGFND5_RMETRIC	1
171	COMPASS_OFS2_X	28.66513	**631**	RNGFND5_SCALING	3
172	COMPASS_OFS2_Y	98.94672	**632**	RNGFND5_STOP_PIN	-1
173	COMPASS_OFS2_Z	-7.05574	**633**	RNGFND5_TYPE	0
174	COMPASS_OFS3_X	0	**634**	RNGFND6_ADDR	0
175	COMPASS_OFS3_Y	0	**635**	RNGFND6_FUNCTION	0
176	COMPASS_OFS3_Z	0	**636**	RNGFND6_GNDCLEAR	10
177	COMPASS_ORIENT	0	**637**	RNGFND6_MAX_CM	700
178	COMPASS_ORIENT2	0	**638**	RNGFND6_MIN_CM	20
179	COMPASS_ORIENT3	0	**639**	RNGFND6_OFFSET	0
180	COMPASS_PMOT_EN	0	**640**	RNGFND6_ORIENT	0
181	COMPASS_PRIMARY	0	**641**	RNGFND6_PIN	-1
182	COMPASS_TYPEMASK	0	**642**	RNGFND6_POS_X	0
183	COMPASS_USE	1	**643**	RNGFND6_POS_Y	0
184	COMPASS_USE2	0	**644**	RNGFND6_POS_Z	0
185	COMPASS_USE3	0	**645**	RNGFND6_PWRRNG	0
186	CRASH_ANGLE	0	**646**	RNGFND6_RMETRIC	1
187	CRUISE_SPEED	1.111076	**647**	RNGFND6_SCALING	3
188	CRUISE_THROTTLE	100	**648**	RNGFND6_STOP_PIN	-1
189	EK2_ABIAS_P_NSE	0.005	**649**	RNGFND6_TYPE	0
190	EK2_ACC_P_NSE	0.6	**650**	RNGFND7_ADDR	0
191	EK2_ALT_M_NSE	3	**651**	RNGFND7_FUNCTION	0
192	EK2_ALT_SOURCE	0	**652**	RNGFND7_GNDCLEAR	10
193	EK2_BCN_DELAY	50	**653**	RNGFND7_MAX_CM	700
194	EK2_BCN_I_GTE	500	**654**	RNGFND7_MIN_CM	20
195	EK2_BCN_M_NSE	1	**655**	RNGFND7_OFFSET	0
196	EK2_CHECK_SCALE	100	**656**	RNGFND7_ORIENT	0
197	EK2_EAS_I_GATE	400	**657**	RNGFND7_PIN	-1
198	EK2_EAS_M_NSE	1.4	**658**	RNGFND7_POS_X	0
199	EK2_ENABLE	1	**659**	RNGFND7_POS_Y	0
200	EK2_EXTNAV_DELAY	10	**660**	RNGFND7_POS_Z	0
201	EK2_FLOW_DELAY	10	**661**	RNGFND7_PWRRNG	0
202	EK2_FLOW_I_GATE	300	**662**	RNGFND7_RMETRIC	1
203	EK2_FLOW_M_NSE	0.25	**663**	RNGFND7_SCALING	3

204	EK2_FLOW_USE	1	**664**	RNGFND7_STOP_PIN	-1
205	EK2_GBIAS_P_NSE	0.0001	**665**	RNGFND7_TYPE	0
206	EK2_GLITCH_RAD	25	**666**	RNGFND8_ADDR	0
207	EK2_GPS_CHECK	31	**667**	RNGFND8_FUNCTION	0
208	EK2_GPS_TYPE	1	**668**	RNGFND8_GNDCLEAR	10
209	EK2_GSCL_P_NSE	0.0005	**669**	RNGFND8_MAX_CM	700
210	EK2_GYRO_P_NSE	0.03	**670**	RNGFND8_MIN_CM	20
211	EK2_HGT_DELAY	60	**671**	RNGFND8_OFFSET	0
212	EK2_HGT_I_GATE	500	**672**	RNGFND8_ORIENT	0
213	EK2_HRT_FILT	2	**673**	RNGFND8_PIN	-1
214	EK2_IMU_MASK	3	**674**	RNGFND8_POS_X	0
215	EK2_LOG_MASK	1	**675**	RNGFND8_POS_Y	0
216	EK2_MAG_CAL	2	**676**	RNGFND8_POS_Z	0
217	EK2_MAG_EF_LIM	50	**677**	RNGFND8_PWRRNG	0
218	EK2_MAG_I_GATE	300	**678**	RNGFND8_RMETRIC	1
219	EK2_MAG_M_NSE	0.05	**679**	RNGFND8_SCALING	3
220	EK2_MAG_MASK	0	**680**	RNGFND8_STOP_PIN	-1
221	EK2_MAGB_P_NSE	0.0001	**681**	RNGFND8_TYPE	0
222	EK2_MAGE_P_NSE	0.001	**682**	RNGFND9_ADDR	0
223	EK2_MAX_FLOW	2.5	**683**	RNGFND9_FUNCTION	0
224	EK2_NOAID_M_NSE	10	**684**	RNGFND9_GNDCLEAR	10
225	EK2_OGN_HGT_MASK	0	**685**	RNGFND9_MAX_CM	700
226	EK2_POS_I_GATE	500	**686**	RNGFND9_MIN_CM	20
227	EK2_POSNE_M_NSE	0.1	**687**	RNGFND9_OFFSET	0
228	EK2_RNG_I_GATE	500	**688**	RNGFND9_ORIENT	0
229	EK2_RNG_M_NSE	0.5	**689**	RNGFND9_PIN	-1
230	EK2_RNG_USE_HGT	-1	**690**	RNGFND9_POS_X	0
231	EK2_RNG_USE_SPD	2	**691**	RNGFND9_POS_Y	0
232	EK2_TAU_OUTPUT	25	**692**	RNGFND9_POS_Z	0
233	EK2_TERR_GRAD	0.1	**693**	RNGFND9_PWRRNG	0
234	EK2_VEL_I_GATE	500	**694**	RNGFND9_RMETRIC	1
235	EK2_VELD_M_NSE	0.7	**695**	RNGFND9_SCALING	3
236	EK2_VELNE_M_NSE	0.1	**696**	RNGFND9_STOP_PIN	-1
237	EK2_WIND_P_NSE	0.1	**697**	RNGFND9_TYPE	0
238	EK2_WIND_PSCALE	0.5	**698**	RNGFNDA_ADDR	0
239	EK2_YAW_I_GATE	300	**699**	RNGFNDA_FUNCTION	0
240	EK2_YAW_M_NSE	0.5	**700**	RNGFNDA_GNDCLEAR	10
241	EK3_ENABLE	0	**701**	RNGFNDA_MAX_CM	700
242	FENCE_ACTION	1	**702**	RNGFNDA_MIN_CM	20
243	FENCE_ENABLE	0	**703**	RNGFNDA_OFFSET	0
244	FENCE_MARGIN	2	**704**	RNGFNDA_ORIENT	0
245	FENCE_RADIUS	300	**705**	RNGFNDA_PIN	-1

246	FENCE_TOTAL	0	**706**	RNGFNDA_POS_X	0
247	FENCE_TYPE	6	**707**	RNGFNDA_POS_Y	0
248	FOLL_ENABLE	0	**708**	RNGFNDA_POS_Z	0
249	FORMAT_VERSION	16	**709**	RNGFNDA_PWRRNG	0
250	FRAME_CLASS	1	**710**	RNGFNDA_RMETRIC	1
251	FRAME_TYPE	0	**711**	RNGFNDA_SCALING	3
252	FS_ACTION	2	**712**	RNGFNDA_STOP_PIN	-1
253	FS_CRASH_CHECK	0	**713**	RNGFNDA_TYPE	0
254	FS_EKF_ACTION	1	**714**	RPM_MAX	100000
255	FS_EKF_THRESH	0.8	**715**	RPM_MIN	10
256	FS_GCS_ENABLE	0	**716**	RPM_MIN_QUAL	0.5
257	FS_OPTIONS	0	**717**	RPM_PIN	54
258	FS_THR_ENABLE	1	**718**	RPM_SCALING	1
259	FS_THR_VALUE	910	**719**	RPM_TYPE	0
260	FS_TIMEOUT	1.5	**720**	RPM2_PIN	-1
261	GCS_PID_MASK	0	**721**	RPM2_SCALING	1
262	GND_ABS_PRESS	97793.36	**722**	RPM2_TYPE	0
263	GND_ABS_PRESS2	0	**723**	RSSI_TYPE	0
264	GND_ABS_PRESS3	0	**724**	RST_SWITCH_CH	0
265	GND_ALT_OFFSET	0	**725**	RTL_SPEED	0
266	GND_EXT_BUS	-1	**726**	SAIL_ENABLE	0
267	GND_FLTR_RNG	0	**727**	SCHED_DEBUG	0
268	GND_PRIMARY	0	**728**	SCHED_LOOP_RATE	50
269	GND_PROBE_EXT	0	**729**	SCR_ENABLE	0
270	GND_TEMP	0	**730**	SERIAL_PASS1	0
271	GPS_AUTO_CONFIG	1	**731**	SERIAL_PASS2	-1
272	GPS_AUTO_SWITCH	3	**732**	SERIAL_PASSTIMO	15
273	GPS_BLEND_MASK	5	**733**	SERIAL0_BAUD	115
274	GPS_BLEND_TC	10	**734**	SERIAL0_PROTOCOL	2
275	GPS_DELAY_MS	0	**735**	SERIAL1_BAUD	57
276	GPS_DELAY_MS2	0	**736**	SERIAL1_OPTIONS	0
277	GPS_GNSS_MODE	0	**737**	SERIAL1_PROTOCOL	1
278	GPS_GNSS_MODE2	77	**738**	SERIAL2_BAUD	57
279	GPS_INJECT_TO	127	**739**	SERIAL2_OPTIONS	0
280	GPS_MIN_DGPS	100	**740**	SERIAL2_PROTOCOL	1
281	GPS_MIN_ELEV	-100	**741**	SERIAL3_BAUD	38
282	GPS_NAVFILTER	8	**742**	SERIAL3_OPTIONS	0
283	GPS_POS1_X	0.185	**743**	SERIAL3_PROTOCOL	5
284	GPS_POS1_Y	0	**744**	SERIAL4_BAUD	115
285	GPS_POS1_Z	-0.175	**745**	SERIAL4_OPTIONS	0
286	GPS_POS2_X	-0.125	**746**	SERIAL4_PROTOCOL	5
287	GPS_POS2_Y	0	**747**	SERIAL5_BAUD	57

288	GPS_POS2_Z	-0.11	**748**	SERIAL5_OPTIONS	0
289	GPS_RATE_MS	200	**749**	SERIAL5_PROTOCOL	-1
290	GPS_RATE_MS2	200	**750**	SERIAL6_BAUD	57
291	GPS_RAW_DATA	0	**751**	SERIAL6_OPTIONS	0
292	GPS_SAVE_CFG	2	**752**	SERIAL6_PROTOCOL	-1
293	GPS_SBAS_MODE	2	**753**	SERIAL7_BAUD	115200
294	GPS_SBP_LOGMASK	-256	**754**	SERIAL7_OPTIONS	0
295	GPS_TYPE	1	**755**	SERIAL7_PROTOCOL	2
296	GPS_TYPE2	5	**756**	SERVO_BLH_DEBUG	0
297	GRIP_ENABLE	0	**757**	SERVO_BLH_MASK	0
298	INITIAL_MODE	0	**758**	SERVO_BLH_OTYPE	0
299	INS_ACC_BODYFIX	2	**759**	SERVO_BLH_POLES	14
300	INS_ACC_ID	2621706	**760**	SERVO_BLH_PORT	0
301	INS_ACC2_ID	2688010	**761**	SERVO_BLH_REMASK	0
302	INS_ACC2OFFS_X	0.161591	**762**	SERVO_BLH_TEST	0
303	INS_ACC2OFFS_Y	-0.03968	**763**	SERVO_BLH_TMOUT	0
304	INS_ACC2OFFS_Z	0.138354	**764**	SERVO_BLH_TRATE	10
305	INS_ACC2SCAL_X	0.991839	**765**	SERVO_RATE	50
306	INS_ACC2SCAL_Y	0.990431	**766**	SERVO_ROB_POSMAX	4095
307	INS_ACC2SCAL_Z	0.981222	**767**	SERVO_ROB_POSMIN	0
308	INS_ACC3_ID	0	**768**	SERVO_SBUS_RATE	50
309	INS_ACC3OFFS_X	0	**769**	SERVO_VOLZ_MASK	0
310	INS_ACC3OFFS_Y	0	**770**	SERVO1_FUNCTION	73
311	INS_ACC3OFFS_Z	0	**771**	SERVO1_MAX	1880
312	INS_ACC3SCAL_X	0	**772**	SERVO1_MIN	1100
313	INS_ACC3SCAL_Y	0	**773**	SERVO1_REVERSED	0
314	INS_ACC3SCAL_Z	0	**774**	SERVO1_TRIM	1500
315	INS_ACCEL_FILTER	10	**775**	SERVO10_FUNCTION	0
316	INS_ACCOFFS_X	-0.08955	**776**	SERVO10_MAX	1900
317	INS_ACCOFFS_Y	0.172145	**777**	SERVO10_MIN	1100
318	INS_ACCOFFS_Z	0.064103	**778**	SERVO10_REVERSED	0
319	INS_ACCSCAL_X	0.997815	**779**	SERVO10_TRIM	1500
320	INS_ACCSCAL_Y	0.999581	**780**	SERVO11_FUNCTION	0
321	INS_ACCSCAL_Z	0.98419	**781**	SERVO11_MAX	1900
322	INS_ENABLE_MASK	127	**782**	SERVO11_MIN	1100
323	INS_FAST_SAMPLE	1	**783**	SERVO11_REVERSED	0
324	INS_GYR_CAL	1	**784**	SERVO11_TRIM	1500
325	INS_GYR_ID	2621706	**785**	SERVO12_FUNCTION	0
326	INS_GYR2_ID	2687754	**786**	SERVO12_MAX	1900
327	INS_GYR2OFFS_X	-0.00299	**787**	SERVO12_MIN	1100
328	INS_GYR2OFFS_Y	0.003449	**788**	SERVO12_REVERSED	0
329	INS_GYR2OFFS_Z	0.001118	**789**	SERVO12_TRIM	1500

330	INS_GYR3_ID	0	**790**	SERVO13_FUNCTION	0
331	INS_GYR3OFFS_X	0	**791**	SERVO13_MAX	1900
332	INS_GYR3OFFS_Y	0	**792**	SERVO13_MIN	1100
333	INS_GYR3OFFS_Z	0	**793**	SERVO13_REVERSED	0
334	INS_GYRO_FILTER	4	**794**	SERVO13_TRIM	1500
335	INS_GYROFFS_X	0.025607	**795**	SERVO14_FUNCTION	0
336	INS_GYROFFS_Y	-0.01907	**796**	SERVO14_MAX	1900
337	INS_GYROFFS_Z	-0.00255	**797**	SERVO14_MIN	1100
338	INS_HNTCH_ENABLE	0	**798**	SERVO14_REVERSED	0
339	INS_LOG_BAT_CNT	1024	**799**	SERVO14_TRIM	1500
340	INS_LOG_BAT_LGCT	32	**800**	SERVO15_FUNCTION	0
341	INS_LOG_BAT_LGIN	20	**801**	SERVO15_MAX	1900
342	INS_LOG_BAT_MASK	0	**802**	SERVO15_MIN	1100
343	INS_LOG_BAT_OPT	0	**803**	SERVO15_REVERSED	0
344	INS_NOTCH_ENABLE	0	**804**	SERVO15_TRIM	1500
345	INS_POS1_X	0.04	**805**	SERVO16_FUNCTION	0
346	INS_POS1_Y	0.05	**806**	SERVO16_MAX	1900
347	INS_POS1_Z	0.045	**807**	SERVO16_MIN	1100
348	INS_POS2_X	0	**808**	SERVO16_REVERSED	0
349	INS_POS2_Y	0	**809**	SERVO16_TRIM	1500
350	INS_POS2_Z	0	**810**	SERVO2_FUNCTION	0
351	INS_POS3_X	0	**811**	SERVO2_MAX	1900
352	INS_POS3_Y	0	**812**	SERVO2_MIN	1100
353	INS_POS3_Z	0	**813**	SERVO2_REVERSED	0
354	INS_STILL_THRESH	0.1	**814**	SERVO2_TRIM	1500
355	INS_TRIM_OPTION	1	**815**	SERVO3_FUNCTION	74
356	INS_USE	1	**816**	SERVO3_MAX	1950
357	INS_USE2	1	**817**	SERVO3_MIN	1100
358	INS_USE3	1	**818**	SERVO3_REVERSED	0
359	LOG_BACKEND_TYPE	1	**819**	SERVO3_TRIM	1500
360	LOG_BITMASK	65535	**820**	SERVO4_FUNCTION	0
361	LOG_DISARMED	1	**821**	SERVO4_MAX	1900
362	LOG_FILE_BUFSIZE	50	**822**	SERVO4_MIN	1100
363	LOG_FILE_DSRMROT	0	**823**	SERVO4_REVERSED	0
364	LOG_FILE_TIMEOUT	5	**824**	SERVO4_TRIM	1500
365	LOG_MAV_BUFSIZE	8	**825**	SERVO5_FUNCTION	0
366	LOG_REPLAY	0	**826**	SERVO5_MAX	1900
367	LOIT_RADIUS	2	**827**	SERVO5_MIN	1100
368	LOIT_SPEED_GAIN	0.5	**828**	SERVO5_REVERSED	0
369	LOIT_TYPE	0	**829**	SERVO5_TRIM	1500
370	MIS_DONE_BEHAVE	0	**830**	SERVO6_FUNCTION	0
371	MIS_OPTIONS	0	**831**	SERVO6_MAX	1900

372	MIS_RESTART	0	**832**	SERVO6_MIN	1100
373	MIS_TOTAL	36	**833**	SERVO6_REVERSED	0
374	MNT_ANGMAX_PAN	4500	**834**	SERVO6_TRIM	1500
375	MNT_ANGMAX_ROL	4500	**835**	SERVO7_FUNCTION	0
376	MNT_ANGMAX_TIL	4500	**836**	SERVO7_MAX	1900
377	MNT_ANGMIN_PAN	-4500	**837**	SERVO7_MIN	1100
378	MNT_ANGMIN_ROL	-4500	**838**	SERVO7_REVERSED	0
379	MNT_ANGMIN_TIL	-4500	**839**	SERVO7_TRIM	1500
380	MNT_DEFLT_MODE	3	**840**	SERVO8_FUNCTION	0
381	MNT_JSTICK_SPD	0	**841**	SERVO8_MAX	1900
382	MNT_LEAD_PTCH	0	**842**	SERVO8_MIN	1100
383	MNT_LEAD_RLL	0	**843**	SERVO8_REVERSED	0
384	MNT_NEUTRAL_X	0	**844**	SERVO8_TRIM	1500
385	MNT_NEUTRAL_Y	0	**845**	SERVO9_FUNCTION	0
386	MNT_NEUTRAL_Z	0	**846**	SERVO9_MAX	1900
387	MNT_RC_IN_PAN	0	**847**	SERVO9_MIN	1100
388	MNT_RC_IN_ROLL	0	**848**	SERVO9_REVERSED	0
389	MNT_RC_IN_TILT	0	**849**	SERVO9_TRIM	1500
390	MNT_RETRACT_X	0	**850**	SIMPLE_TYPE	0
391	MNT_RETRACT_Y	0	**851**	SPEED_MAX	0
392	MNT_RETRACT_Z	0	**852**	SPRAY_ENABLE	0
393	MNT_STAB_PAN	0	**853**	SR0_ADSB	0
394	MNT_STAB_ROLL	0	**854**	SR0_EXT_STAT	2
395	MNT_STAB_TILT	0	**855**	SR0_EXTRA1	4
396	MNT_TYPE	0	**856**	SR0_EXTRA2	4
397	MODE_CH	6	**857**	SR0_EXTRA3	2
398	MODE1	10	**858**	SR0_PARAMS	10
399	MODE2	4	**859**	SR0_POSITION	2
400	MODE3	5	**860**	SR0_RAW_CTRL	1
401	MODE4	0	**861**	SR0_RAW_SENS	2
402	MODE5	0	**862**	SR0_RC_CHAN	2
403	MODE6	3	**863**	SR1_ADSB	0
404	MOT_PWM_FREQ	16	**864**	SR1_EXT_STAT	2
405	MOT_PWM_TYPE	0	**865**	SR1_EXTRA1	4
406	MOT_SAFE_DISARM	0	**866**	SR1_EXTRA2	4
407	MOT_SLEWRATE	100	**867**	SR1_EXTRA3	2
408	MOT_SPD_SCA_BASE	1	**868**	SR1_PARAMS	10
409	MOT_THR_MAX	100	**869**	SR1_POSITION	2
410	MOT_THR_MIN	4	**870**	SR1_RAW_CTRL	1
411	MOT_THST_EXPO	0	**871**	SR1_RAW_SENS	2
412	MOT_VEC_THR_BASE	0	**872**	SR1_RC_CHAN	2
413	NAVL1_DAMPING	0.75	**873**	SR2_ADSB	0

414	NAVL1_PERIOD	11	**874**	SR2_EXT_STAT	1
415	NAVL1_XTRACK_I	0.02	**875**	SR2_EXTRA1	1
416	NTF_BUZZ_ENABLE	1	**876**	SR2_EXTRA2	1
417	NTF_BUZZ_ON_LVL	1	**877**	SR2_EXTRA3	1
418	NTF_BUZZ_PIN	0	**878**	SR2_PARAMS	10
419	NTF_BUZZ_VOLUME	100	**879**	SR2_POSITION	1
420	NTF_DISPLAY_TYPE	0	**880**	SR2_RAW_CTRL	1
421	NTF_LED_BRIGHT	3	**881**	SR2_RAW_SENS	1
422	NTF_LED_OVERRIDE	0	**882**	SR2_RC_CHAN	1
423	NTF_LED_TYPES	199	**883**	SR3_ADSB	0
424	NTF_OREO_THEME	0	**884**	SR3_EXT_STAT	2
425	OA_TYPE	0	**885**	SR3_EXTRA1	4
426	PILOT_STEER_TYPE	0	**886**	SR3_EXTRA2	4
427	PRX_IGN_ANG1	0	**887**	SR3_EXTRA3	2
428	PRX_IGN_ANG2	0	**888**	SR3_PARAMS	10
429	PRX_IGN_ANG3	0	**889**	SR3_POSITION	2
430	PRX_IGN_ANG4	0	**890**	SR3_RAW_CTRL	1
431	PRX_IGN_ANG5	0	**891**	SR3_RAW_SENS	2
432	PRX_IGN_ANG6	0	**892**	SR3_RC_CHAN	2
433	PRX_IGN_WID1	0	**893**	SRTL_ACCURACY	2
434	PRX_IGN_WID2	0	**894**	SRTL_POINTS	300
435	PRX_IGN_WID3	0	**895**	STAT_BOOTCNT	17
436	PRX_IGN_WID4	0	**896**	STAT_FLTTIME	2257
437	PRX_IGN_WID5	0	**897**	STAT_RESET	160496500
438	PRX_IGN_WID6	0	**898**	STAT_RUNTIME	156158
439	PRX_ORIENT	0	**899**	STICK_MIXING	0
440	PRX_TYPE	0	**900**	SYSID_ENFORCE	0
441	PRX_YAW_CORR	0	**901**	SYSID_MYGCS	255
442	RALLY_INCL_HOME	1	**902**	SYSID_THISMAV	1
443	RALLY_LIMIT_KM	0.5	**903**	TELEM_DELAY	0
444	RALLY_TOTAL	0	**904**	TURN_MAX_G	0.6
445	RC_OPTIONS	0	**905**	TURN_RADIUS	0.1
446	RC_OVERRIDE_TIME	3	**906**	VISO_ORIENT	0
447	RC1_DZ	0	**907**	VISO_POS_X	0
448	RC1_MAX	1901	**908**	VISO_POS_Y	0
449	RC1_MIN	1099	**909**	VISO_POS_Z	0
450	RC1_OPTION	0	**910**	VISO_TYPE	0
451	RC1_REVERSED	0	**911**	WENC_TYPE	0
452	RC1_TRIM	1099	**912**	WNDVN_TYPE	0
453	RC10_DZ	0	**913**	WP_OVERSHOOT	0.03
454	RC10_MAX	2000	**914**	WP_PIVOT_ANGLE	60
455	RC10_MIN	1000	**915**	WP_PIVOT_RATE	90

456	RC10_OPTION	0	**916**	WP_RADIUS	0.03
457	RC10_REVERSED	0	**917**	WP_SPEED	1
458	RC10_TRIM	1500	**918**	WP_SPEED_MIN	0
459	RC11_DZ	0	**919**	WRC_ENABLE	0
460	RC11_MAX	2000			

Appendix B

Python Code

B.1 inline_pair_UGV01.py

This script generates a tab-delimited *.waypoints file compatible with Mission Planner. Using the known locations of the orchard row end posts and the locations of the test track GCPs, the script can generate waypoints for a mission in a given direction at a given spacing interval for waypoints. Spacing options are 50% of a pass, 25% of a pass, or any fixed distance in feet. This script was originally authored by Dr. H.J. Sommer and was later modified by Michael Pagan.

```
""" inline_pair_UGV01.py - main for centerline between pair of rows """
__author__ = "HJSIII, 21.01.27" # Modified by Michael Pagan

# ########################################################################
# import
from math import *
import numpy as np
import csv

# ########################################################################
# local constants
d2r = pi / 180.0

# 1 deg lat = 364813 feet,
# 1 deg lon = cos(lat)*d2f_lat, MATLAB spherical Earth model
d2f_lat = 364813.0

# mission parameters
overshoot = 15.0          #overshoot at begining/end of rows [ft]
spacing = 5.0             #in-line spacing between waypoints (fixed) [ft]
h = 0                     #AGL [ft]
spacing_option = '25%'            #50% spacing = '50%',
                                #25% spacing = '25%',
                          #fixed = 'fixed'

# ########################################################################
# open CSV file to write text
fn_csv = 'mission_plan.waypoints'
fid_csv = open( fn_csv, 'w' )
```

```
# write header with new line at end
header = 'QGC WPL 110\n'
fid_csv.write( header )
#write index 0 line (home position)
fid_csv.write( '%5.0f\t' % (0) )        # index
fid_csv.write( '0\t3\t16\t0\t0\t0\t0\t0\t0\t0\t1\n' )
          # current_wp, coord_frame, command, param1, param2, param3,
          # param4, lat, long, AGL, autocontinue

# ####################################################################
# direction = 'FOR'
# forward - apple rows Nlat, NLon, Slat, Slon - TEST Wlat Wlon Elat Elon
# direction = 'REV'      # reverse
# select  block  row_a  row_b  direction
# comment out either test track or apple rows

              #test track#
        #block  row_a  row_b  direction#
#babd = [ 'TEST',  '1',  '1',  'FOR',
         #'TEST',  '2',  '2',  'REV',
         #'TEST',  '1',  '1',  'FOR',
         #'TEST',  '2',  '2',  'REV'  ]

              ##apple rows#
        ##block  row_a  row_b  direction##
babd = [ 'A3',  'AA',  'AB',  'FOR',
         'A3',  'AB',  'AC',  'REV',
         'A3',  'AC',  'AD',   'FOR',
         'A3',  'AD',   'BA',  'REV',
         'A3',  'BA',   'BB',  'FOR',
         'A3',  'BB',   'CA',  'REV',
         'A3',  'CA',   'CB',  'FOR']

# size
n_babd = len( babd )
n_babd = int( n_babd / 4 )

# process one pass at a time
n_pass = 1
for i_babd in range( n_babd ):

# rip
    block     = babd[ i_babd*4     ]    #block, row a, row b, direction
    row_a     = babd[ i_babd*4 + 1 ]
    row_b     = babd[ i_babd*4 + 2 ]
    direction = babd[ i_babd*4 + 3 ]

# read CSV file with lat-lon for posts and find N-S ends
# CSV contains - block  row  N_lat  N_lon  S_lat  S_lon
    nlat_a = 0
    nlat_b = 0
    fn_posts = '200305 rows cut.csv'
    with open( fn_posts, newline='') as csvfile:
        reader = csv.reader( csvfile )
        for line in reader:
            if line[0] == block and line[1] == row_a:
                nlat_a = float( line[2] )
                nlon_a = float( line[3] )
                slat_a= float( line[4] )
                slon_a = float( line[5] )
            if line[0] == block and line[1] == row_b:
                nlat_b = float( line[2] )
                nlon_b = float( line[3] )
                slat_b = float( line[4] )
                slon_b = float( line[5] )

    if nlat_a == 0:
        print( '\nWARNING - Block', block, 'Row', row_a, 'not found\n'  )
    if nlat_b == 0:
        print( '\nWARNING - Block', block, 'Row', row_b, 'not found\n'  )

# read CSV straight into numpy array like Matlab "load"
# from numpy import genfromtxt
# my_data = genfromtxt('my_file.csv', delimiter=',')

# normally fly missions N to S for apple, W to E for TEST
    lat1 = ( nlat_a + nlat_b ) / 2
    lon1 = ( nlon_a + nlon_b ) / 2
    lat2 = ( slat_a + slat_b ) / 2
    lon2 = ( slon_a + slon_b ) / 2

# reverse direction
    if direction == 'REV':
        lat1, lat2 = lat2, lat1
        lon1, lon2 = lon2, lon1

# overall path - x East, y North
    del_lat = lat2 - lat1
    del_lon = lon2 - lon1

    d2f_lon = cos( lat1*d2r ) * d2f_lat
    del_x = del_lon * d2f_lon
    del_y = del_lat * d2f_lat

    distance = sqrt( del_x*del_x + del_y*del_y )
    theta = atan2( del_y, del_x )
    theta_deg = theta / d2r
    heading_path = fmod( (90 - theta_deg + 360), 360 )

    # Spacing
    if spacing_option == 'fixed' and spacing > 0:
        xloc = np.arange( -overshoot,    #for fixed-distance waypoint spacing
      ((distance+2*overshoot)%spacing+distance+overshoot), spacing)
    elif spacing_option == '25%':
        xloc = np.array( [ -overshoot,   # for 25% waypoint spacing
      (distance+2*overshoot)/4-overshoot,2*(distance+2*overshoot)/4-overshoot,
      3*(distance+2*overshoot)/4 -overshoot, distance+overshoot ] )
    else:
        xloc = np.array( [ -overshoot,  (distance+2*overshoot)/2-overshoot,
                           distance+overshoot ] ) #for 50% waypoint spacing

    yloc = np.full( xloc.size, 0 )
    head_plan = np.full( xloc.size, heading_path )

# rotate into global and convert to lat-lon
    xyloc = np.vstack( ( xloc, yloc ) )
    Amat = np.array( [ ( cos(theta), -sin(theta) ) ,
                       ( sin(theta),  cos(theta) ) ] )
    xyglo = Amat @ xyloc

    lat_plan = lat1 + xyglo[1]/d2f_lat
    lon_plan = lon1 + xyglo[0]/d2f_lon
    n = len( lat_plan )

# flat AGL
    agl_plan = np.full( lat_plan.size, h )

# save lat-lon for each waypoint - tab delimited
    n = len( lat_plan )
    for i in range(n):
        i_pass = n_pass + i
        fid_csv.write( '%5.0f\t' % (i_pass) )        # index
        fid_csv.write( '0\t3\t16\t0\t0\t0\t0\t' )  # current_wp,
       # coord_frame, command, param1, param2, param3, param4
        fid_csv.write( '%12.7f\t' % lat_plan[i] )  # latitude
        fid_csv.write( '%12.7f\t' % lon_plan[i] )  # longitude
        fid_csv.write( '%6.2f\t'  % agl_plan[i] )  # AGL
        fid_csv.write( '1\n' )                    # autocontinue and new line
        print( i_pass )

# finished with current pass
    n_pass = n_pass + n

# finished with all passes - close file
fid_csv.close()

# bottom - inline_pair_UGV01
```

B.2 mission_update.py

This script uses MAVROS to update the mission on Pixhawk. This code was authored by Michael Pagan and contains contributions from Dr. Sommer's inline_pair_UGV01.py code.

```
__author__ = "MAP | HJSIII | 29 Jan 2021"

#Script intervenes during active mission to:
    #1)Generate new mission  2)Activate HOLD mode    3)Clear current mission
    #4)Download new mission    5)Reset home position   6)Reactivate AUTO mode
# IMPORT LIBRARIES #####################################################
import rospy                            #ROSS
import time                             #for sleeps
from math import *                      #for waypoint generation
import numpy as np                      #^
import csv                              # read file containing row coordinates
from std_msgs.msg import String         #for sending MAVROS messages
from sensor_msgs.msg import NavSatFix   #^
from mavros_msgs.msg import *           #^
from mavros_msgs.srv import *           #^

# MISSION SETUP ####################################################
#####LOCAL CONSTANTS#####
d2r = pi / 180.0    #degrees to radians
d2f_lat = 364813.0  #1 deg lat = 364813 feet, 1 deg lon = cos(lat)*d2f_lat
                      #MATLAB spherical Earth model

#####MISSION PARAMETERS#####
overshoot = 15.0          #overshoot at begining/end of rows [ft]
spacing = 5.0             #fixed in-line spacing between waypoints [ft]
h = 0                     #AGL [ft]
spacing_option = '50%'          # 50% spacing = '50%', 25% spacing = '25%'
                                 #fixed = 'fixed'

#reference '200305 rows cut.csv' to plan row passes
#- file must be in current directory
#direction = 'FOR'      #forward - apple rows Nlat, NLon, Slat, Slon
                                #- TEST Wlat Wlon Elat Elon
direction = 'REV'       #reverse

              #test track#
        #block  row_a  row_b  direction#
babd = [ 'TEST',  '1',  '1',  'FOR',
         'TEST',  '2',  '2',  'REV',
         'TEST',  '1',  '1',  'FOR',
         'TEST',  '2',  '2',  'REV'  ]

              #apple rows#
        #block  row_a  row_b  direction#
babd = [ 'A1',  'B',  'C',  'FOR',
         'A1',  'C',  'D',  'REV',
         'A1',  'EE',  'F',   'FOR',
         'A1',  'F',   'FF',  'REV',
         'A1',  'FF',   'G',  'FOR',
         'A1',  'G',   'GG',  'REV']

n_babd = len( babd )      #find length of mission array
n_babd = int( n_babd / 4 )#divide by number of columns to find number of passes

# CHANGE MODE ##############################################################
#####HOLD MODE#####
def activate_hold():
    try:
        HoldService = rospy.ServiceProxy('mavros/set_mode', SetMode)
        HoldService(base_mode=0, custom_mode = 'HOLD') #call MAVROS set_mode
                                                #service to set mode to HOLD
        if HoldService.call(base_mode=0, custom_mode = 'HOLD').mode_sent:
            print ("HOLD mode activated") #check that the mode_sent message
                                        #is 'true' and print verification
        else:
            print("unable to activate HOLD mode")
    except rospy.ServiceException as exc:
        print ("Failed to call SetMode service for HOLD: " + str(exc))
            #print error message if service call fails

#####AUTO MODE#####
def reactivate_auto():
    try:
        AutoService = rospy.ServiceProxy('mavros/set_mode', SetMode)
        AutoService(base_mode=0, custom_mode = 'AUTO') #call MAVROS set_mode
                                                 #service to set mode to AUTO
        if AutoService.call(base_mode=0, custom_mode = 'AUTO').mode_sent:
            print ("AUTO mode reactivated") #check that the mode_sent message
                                           #is 'true' and print verification
        else:
            print ("unable to activate AUTO, keeping HOLD mode")
    except rospy.ServiceException as exc:
        print ("Failed to call SetMode service for AUTO: " + str(exc))
                           #print error message if service call fails

# RESET MISSION ############################################################
#####CLEAR OLD WAYPOINTS#####
def clear_mission():
    try:
        ClearService = rospy.ServiceProxy('mavros/mission/clear', WaypointClear)
        ClearService()    #call MAVROS clear service to clear waypoints
        if ClearService.call().success:
            print ("waypoint list cleared") #check that the success message
                                        #is 'true' and print verification
        else:
            print("unable to clear waypoint list")
    except rospy.ServiceException as exc:
        print ("Failed to call WaypointClear service: " + str(exc))
                     #print error message if service call fails
        return False

#####RESET CURRENT WAYPOINT#####
def restart_wp_sequence():
    try:
        SequenceService = rospy.ServiceProxy('mavros/mission/set_current',
                                    WaypointSetCurrent)
        SequenceService(1) #call MAVROS set_current service to set
                          #current waypoint to 1
        if (SequenceService.call(1).success):
            print ("waypoint sequence restarted") #check that the success
                                  #message is 'true' and print verification
        else:
            print("unable to restart waypoint sequence")
    except rospy.ServiceException as exc:
        print ("Failed to call WaypointSetCurrent service: " + str(exc))
       #print error message if service call fails

#####RESET HOME POSITION#####
def current_GPS_home():
    try:
        HomeService = rospy.ServiceProxy('/mavros/cmd/set_home', CommandHome)
        HomeService(current_gps = 1, yaw=0, latitude=0, longitude=0, altitude=0)
          #call MAVROS CommandHome service to set current location as home
        if (HomeService.call(current_gps = 1, yaw=0, latitude=0, longitude=0,
                           altitude=0).success):
            print("home position set to current location")
        else:
            print("home position not set")
    except rospy.ServiceException as exc:
        print ("Failed to call CommandHome service: " + str(exc))
       #print error message if service call fails

# CREATE MISSION ##########################################################
def create_waypoint():
    wl = [] #create waypoint list (wl)

    #####HOME POSITION PLACEHOLDER#####
    #waypoint of index = 0 is home location i.e. not part of the mission
    wp = Waypoint() #create object instance 'wp' of 'Waypoint'
                     #class to store each waypoint's data
    wp.frame = 3
    wp.command = 16
    wp.is_current = False
    wp.autocontinue = True
    wp.param1 = 0
    wp.param2 = 0
    wp.param3 = 0
    wp.param4 = 0
    wp.x_lat = 0
    wp.y_long = 0
    wp.z_alt = 0
```

```
wl.append(wp) #add home position placeholder to waypoint list (wl)

#####GENERATE WAYPOINTS#####
n_pass = 0
for i_babd in range( n_babd ): #loop to process one pass at a time
    #rip
    block     = babd[ i_babd*4     ]
    row_a     = babd[ i_babd*4 + 1 ]
    row_b     = babd[ i_babd*4 + 2 ]
    direction = babd[ i_babd*4 + 3 ]

    #read CSV file with lat-lon for posts and find N-S ends
    #CSV contains - block  row  N_lat  N_lon  S_lat  S_lon
    nlat_a = 0
    nlat_b = 0
    fn_posts = '200305 rows cut.csv'
    with open( fn_posts, newline='') as csvfile:
        reader = csv.reader( csvfile )
        for line in reader:
            if line[0] == block and line[1] == row_a:
                nlat_a = float( line[2] )
                nlon_a = float( line[3] )
                slat_a= float( line[4] )
                slon_a = float( line[5] )
            if line[0] == block and line[1] == row_b:
                nlat_b = float( line[2] )
                nlon_b = float( line[3] )
                slat_b = float( line[4] )
                slon_b = float( line[5] )
    if nlat_a == 0:
        print( '\nWARNING - Block', block, 'Row', row_a, 'not found\n'  )
    if nlat_b == 0:
        print( '\nWARNING - Block', block, 'Row', row_b, 'not found\n'  )

    #normally fly missions N to S for apple, W to E for TEST
    lat1 = ( nlat_a + nlat_b ) / 2
    lon1 = ( nlon_a + nlon_b ) / 2
    lat2 = ( slat_a + slat_b ) / 2
    lon2 = ( slon_a + slon_b ) / 2

    #reverse direction
    if direction == 'REV':
        lat1, lat2 = lat2, lat1
        lon1, lon2 = lon2, lon1

    #overall path - x East, y North
    del_lat = lat2 - lat1
    del_lon = lon2 - lon1

    d2f_lon = cos( lat1*d2r ) * d2f_lat
    del_x = del_lon * d2f_lon
    del_y = del_lat * d2f_lat

    distance = sqrt( del_x*del_x + del_y*del_y )
```

```
  theta = atan2( del_y, del_x )
  theta_deg = theta / d2r
  heading_path = fmod( (90 - theta_deg + 360), 360 )

  #plan path in local coordinates - +xloc forward, +yloc left,
#zero at first point of interest (POI)
  if spacing_option == 'fixed' and spacing > 0:
      xloc = np.arange(
         -overshoot,
         ((distance+2*overshoot)%spacing+distance+overshoot), spacing)
             #for fixed-distance waypoint spacing
  elif spacing_option == '25%':
      xloc = np.array( [ -overshoot,  (distance+2*overshoot)/4-overshoot,
                         2*(distance+2*overshoot)/4-overshoot,
                           3*(distance+2*overshoot)/4 -overshoot,
                          distance+overshoot ] ) # 25% waypoint spacing
  else:
      xloc = np.array( [ -overshoot,  (distance+2*overshoot)/2-overshoot,
                         distance+overshoot ] ) #for 50% waypoint spacing

  yloc = np.full( xloc.size, 0 )
  head_plan = np.full( xloc.size, heading_path )

  #rotate into global and convert to lat-lon
  xyloc = np.vstack( ( xloc, yloc ) )
  Amat = np.array( [ ( cos(theta), -sin(theta) ) ,
                     ( sin(theta),  cos(theta) ) ] )
  xyglo = Amat @ xyloc

  lat_plan = lat1 + xyglo[1]/d2f_lat
  lon_plan = lon1 + xyglo[0]/d2f_lon

  #flat AGL
  agl_plan = np.full( lat_plan.size, h )

  #save lat-lon for each waypoint in current pass
  n = len( lat_plan )
  for i in range(n):
      i_pass = n_pass + i
      wp = Waypoint() #reset the object 'wp' to store new waypoint data
      wp.frame = 3
      wp.command = 16
      wp.is_current = False
      wp.autocontinue = True
      wp.param1 = 0
      wp.param2 = 0
      wp.param3 = 0
      wp.param4 = 0
      wp.x_lat = lat_plan[i]
      wp.y_long = lon_plan[i]
      wp.z_alt = agl_plan[i]
      wl.append(wp) #add waypoint to waypoint list (wl)
#finished with current pass
  n_pass = n_pass + n
```

```
    #finished with all passes
    print ("finished generating new waypoints")
    print("downloading new mission...")

    #####PUSH NEW WAYPOINT LIST TO PIXHAWK#####
    try:
        PushService = rospy.ServiceProxy('mavros/mission/push', WaypointPush,
                                persistent=True)
        PushService(start_index=0, waypoints=wl) #call MAVROS service to
                                              #push new waypoint list
        if PushService.call(start_index=0, waypoints=wl).success:
            print ("new mission downloaded") #check that the success message is
                                           #'true' and print verification
        else:
            print("MISSION download ERROR. CHECK MAIN MAVROS TERMINAL.")

    except rospy.ServiceException as exc:
        print ("Failed to call WaypointPush service: " + str(exc))
      #print error message if service call fails

# CALL FUNCTIONS #########################################################
activate_hold()           #switch to HOLD mode
clear_mission()     #clear the waypoint list
create_waypoint()   #delete old mission, download new mission
restart_wp_sequence()     #assign first waypoint as current
current_GPS_home()  #set HOME to current location
time.sleep(1)       #allow all messages to be accepted by Pixhawk

ans = input("Reactivate AUTO mode to begin mission? (y/n): ")
if ans == 'y':
    reactivate_auto()     #switch to AUTO mode, begin mission
else:
    print("HOLD mode maintained")

# bottom – mission_update.py
```

B.3 backup.py

This script works in tandem with the Arduino script HCSR04.ino to achieve simple object avoidance with ROS and the Pixhawk. The script creates a node to subscribe to the range messages published by the Arduino. It checks the range and published R/C override messages if an object is too close.

```
import rospy
from std_msgs.msg import String
from mavros_msgs.msg import OverrideRCIn
from sensor_msgs.msg import Range

def callback(msg):
    distance = msg.range #define distance as range component of Range message
                         #on ultrasound topic
    msg = OverrideRCIn() #redefine msg as OverrideRCIn message published
                         #to the override topic
    print(distance)      #print the distance measured by the sensor
    if (distance < 30):  #if a distance <30 cm is read, override RC
        msg.channels = (0, 1500, 1300, 0, 0, 0, 0, 0)
                        #ch2 in ArduPilot rover is steering, ch3 is throttle
    else:
        msg.channels = (0, 0, 0, 0, 0, 0, 0, 0) #don't override RC if >30cm
    pub.publish(msg) #publish OverrideRCIn message
    rospy.loginfo(msg) #print message to screen

rospy.init_node('ultrasonic_value') #initiate node for publisher/subscriber

sub = rospy.Subscriber('/ultrasound', Range, callback) #subscribe to the
                            #"ultrasound" topic created by Arduino. The message
                            #type of this topic is "Range"
pub = rospy.Publisher('mavros/rc/override', OverrideRCIn, queue_size = 10)
                        #publish to the "override" topic with message type
                        #"OverrideRCIn" while limiting the queue of messages
                        #not yet received by the subcriber to 10

rospy.spin() #keep nodes running until they have been shutdown

# bottom - backup.py
```

Appendix C

Arduino Code

C.1 HCSR04

This script is used to publish range messages from an Arduino HCSR04 ultrasonic sensor. It calculates the range based on delay in ultrasonic pulse and publishes the range to a self-established topic. This script works in tandem with the backup.py script.

```
/*
 * Michael Pagan
 * 2.5.2021
 * Sketch to report distance values from ultrasonic sensor
*/

#include <ros.h>
#include <ros/time.h>
#include <sensor_msgs/Range.h>

ros::NodeHandle  nh;
sensor_msgs::Range range_msg;
ros::Publisher pub_range( "ultrasound", &range_msg); //name of topic that messages
are published to

char frameid[] = "ultrasound";

//define pins
const int trigPin = 5;
const int echoPin = 6;

//define variable types
long pulseLength;
int dist; //quantize distances to integers
int distLast; //used to filter out unreasonable distances

void setup()
{
  pinMode(trigPin, OUTPUT); //set digital trigpin as an output
  pinMode(echoPin, INPUT); //set digital echopin as an input

  nh.initNode();
  nh.advertise(pub_range);

  range_msg.radiation_type = sensor_msgs::Range::ULTRASOUND;
  range_msg.header.frame_id =  frameid;
  range_msg.field_of_view = 0.1;  // fake
  range_msg.min_range = 0.0; // cm
  range_msg.max_range = 400; // cm

  digitalWrite(trigPin, LOW);
  //Serial.begin(57600); %
}

void loop(){
  //output ultrasonic burst @ 40000Hz for 10 microsends
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  pulseLength = pulseIn(echoPin, HIGH); //times the duration of HIGH pulse received
  dist = pulseLength * 0.034 / 2; //calculate distance based on speed of sound =
0.034 cm/microsec

  if (400 > dist) {
    range_msg.range = dist;
    range_msg.header.stamp = nh.now();
    pub_range.publish(&range_msg);
    distLast = dist;
    //Serial.println(dist);
  }
  else if ((400 < dist) || (dist-distLast > 100)) {
    range_msg.range = distLast;
    range_msg.header.stamp = nh.now();
    pub_range.publish(&range_msg);
    //Serial.print(distLast);
  }
  nh.spinOnce();
}

//end HCSR04.ino
```

BIBLIOGRAPHY

[1] “Other Global Navigation Satellite Systems (GNSS),” GPS.gov [Online]. Available: https://www.gps.gov/systems/gnss/. [Accessed: 04-Jan-2021].

[2] US Air Force, 2017, “GPS Accuracy,” GPS.gov [Online]. Available: https://www.gps.gov/systems/gps/performance/accuracy/. [Accessed: 04-Jan-2021].

[3] Sickle, J. Van, 2020, “Two Types of Observables,” GEOG 862 GPS GNSS Geospatial Prof. [Online]. Available: https://www.e-education.psu.edu/geog862/node/1752. [Accessed: 04-Jan-2021].

[4] Sickle, J. Van, 2020, “The One-Percent Rule of Thumb,” GEOG 862 GPS GNSS Geospatial Prof. [Online]. Available: https://www.e-education.psu.edu/geog862/node/1760. [Accessed: 04-Jan-2021].

[5] Sickle, J. Van, 2020, “Real-Time Kinematic and Differential GPS,” GEOG 862 GPS GNSS Geospatial Prof. [Online]. Available: https://www.e-education.psu.edu/geog862/node/1828. [Accessed: 04-Jan-2021].

[6] Gan-Mor, S., Clark, R. L., and Upchurch, B. L., 2007, “Implement Lateral Position Accuracy under RTK-GPS Tractor Guidance,” Comput. Electron. Agric., **59**(1–2), pp. 31–38.

[7] Grisetti, G., Kummerle, R., Stachniss, C., and Burgard, W., 2010, “A Tutorial on Graph-Based SLAM,” IEEE Intell. Transp. Syst. Mag., **2**(4), pp. 31–43.

[8] NOAA, 2020, “What Is LIDAR?,” Natl. Ocean Serv. Website [Online]. Available: https://oceanservice.noaa.gov/facts/lidar.html. [Accessed: 13-Apr-2020].

[9] Davies, E. R., 2004, “The Nature of Vision,” *Machine Vision: Theory, Algorithms,*

Practicalities, Elsevier, pp. 2–10.

[10] Rawashdeh, N. A., and Jasim, H. T., 2013, “Mult-Sensor Input Path Planning for an Autonomous Ground Vehicle,” *2013 9th International Symposium on Mechatronics and Its Applications, ISMA 2013*.

[11] Anand, K., and R., G., 2019, “An Autonomous UAV for Pesticide Spraying,” Int. J. Trend Sci. Res. Dev., **Volume-3**(Issue-3), pp. 986–990.

[12] Bochtis, D. D., Vougioukas, S. G., and Griepentrog, H. W., 2009, “A Mission Planner for an Autonomous Tractor,” Trans. ASABE, **52**(5), pp. 1429–1440.

[13] Bochtis, D., Griepentrog, H. W., Vougioukas, S., Busato, P., Berruto, R., and Zhou, K., 2015, “Route Planning for Orchard Operations,” Comput. Electron. Agric., **113**, pp. 51–60.

[14] Gomez-Gil, J., Ruiz-Gonzalez, R., Alonso-Garcia, S., and Gomez-Gil, F. J., 2013, “A Kalman Filter Implementation for Precision Improvement in Low-Cost GPS Positioning of Tractors,” Sensors (Switzerland), **13**(11), pp. 15307–15323.

[15] Moorehead, S. S. J., Wellington, C. K. C., Gilmore, B. J., and Vallespi, C., 2012, “Automating Orchards: A System of Autonomous Tractors for Orchard Maintenance,” Proc. IEEE Int. Conf. Intell. Robot. Syst. Work. Agric. Robot., (January), p. 632.

[16] Barawid, O. C., Mizushima, A., Ishii, K., and Noguchi, N., 2007, “Development of an Autonomous Navigation System Using a Two-Dimensional Laser Scanner in an Orchard Application,” Biosyst. Eng., **96**(2), pp. 139–149.

[17] Hamner, B., Singh, S., and Bergerman, M., 2010, “Improving Orchard Efficiency with Autonomous Utility Vehicles,” *American Society of Agricultural and Biological Engineers Annual International Meeting 2010, ASABE 2010*, pp. 4670–4685.

[18] Bayar, G., Bergerman, M., Koku, A. B., and Konukseven, E. I., 2015, “Localization and Control of an Autonomous Orchard Vehicle,” Comput. Electron. Agric., **115**, pp. 118–128.

[19] Radcliffe, J., Cox, J., and Bulanon, D. M., 2018, “Machine Vision for Orchard Navigation,” Comput. Ind., **98**, pp. 165–171.

[20] 2008, “R/C and Robotics Software for Linux/PXA255/PXA270” [Online]. Available: http://www.pabr.org/pxarc/doc/pxarc.en.html. [Accessed: 21-Jan-2021].

[21] ArduPilot Dev Team, 2020, “Rover Home” [Online]. Available: https://ardupilot.org/rover/. [Accessed: 28-Aug-2020].

[22] Flores, J., Solovey, G., and Gil, S., 2003, “Flow of Sand and a Variable Mass Atwood Machine,” Am. J. Phys., **71**(7), pp. 715–720.

[23] U-blox, 2018, “ZED-F9P Datasheet” [Online]. Available: https://cdn.sparkfun.com/assets/8/3/2/b/8/ZED-F9P_Data_Sheet.pdf. [Accessed: 05-Nov-2020].

[24] Nathan Seidle, 2018, “GPS-RTK2 Hookup Guide,” SparkFun [Online]. Available: https://learn.sparkfun.com/tutorials/gps-rtk2-hookup-guide. [Accessed: 05-Nov-2020].

[25] Nathan Seidle, 2017, “GPS-RTK Hookup Guide,” SparkFun [Online]. Available: https://learn.sparkfun.com/tutorials/gps-rtk-hookup-guide. [Accessed: 05-Nov-2020].

[26] Nathan Seidle, 2020, “Setting up a Rover Base RTK System,” SparkFun [Online]. Available: https://learn.sparkfun.com/tutorials/setting-up-a-rover-base-rtk-system. [Accessed: 05-Nov-2020].

[27] Nathan Seidle, 2020, “How to Build a DIY GNSS Reference Station,” SparkFun [Online]. Available: https://learn.sparkfun.com/tutorials/how-to-build-a-diy-gnss-reference-station.

[Accessed: 05-Nov-2020].

[28] FieldBee, 2020, “FieldBee RTK GPS System” [Online]. Available: https://www.fieldbee.com/product/rtk-gps-system/. [Accessed: 24-Feb-2021].

[29] MAVLINK, 2021, “MAVLink Developer Guide” [Online]. Available: https://mavlink.io/en/. [Accessed: 05-Jan-2021].

[30] RFDesign, 2013, “RFD900 Radio Modem Data Sheet” [Online]. Available: https://files.rfdesign.com.au/Files/documents/RFD900 DataSheet.pdf. [Accessed: 12-Nov-2020].

[31] U-blox, 2021, “U-Center” [Online]. Available: https://www.u-blox.com/en/product/u-center. [Accessed: 05-Nov-2020].

[32] Support, 2017, “RTCM 3 Message List,” SNIP [Online]. Available: https://www.use-snip.com/kb/knowledge-base/rtcm-3-message-list/?gclid=CjwKCAiA4rGCBhAQEiwAelVti6WGjcnnOVNSCkZl5XIZvH3vah5-5S0koG9qIuKe343g3JOsZdqOUxoC1DsQAvD_BwE. [Accessed: 10-Oct-2021].

[33] Inc., S. T., 2007, “NMEA Reference Manual” [Online]. Available: https://www.sparkfun.com/datasheets/GPS/NMEA Reference Manual-Rev2.1-Dec07.pdf. [Accessed: 05-Nov-2020].

[34] MathWorks, 2020, “EarthRadius” [Online]. Available: https://www.mathworks.com/help/map/ref/earthradius.html. [Accessed: 11-May-2020].

[35] ROS, 2021, “RxmRAWX Message” [Online]. Available: http://docs.ros.org/en/kinetic/api/ublox_msgs/html/msg/RxmRAWX.html.

[36] Choy, S., 2018, “GNSS Precise Point Positioning” [Online]. Available: https://www.unoosa.org/documents/pdf/icg/2018/ait-gnss/16_PPP.pdf.

[37] Rtkexplorer, 2021, “RTKLIB Code: Windows Executables” [Online]. Available: http://rtkexplorer.com/downloads/rtklib-code/.

[38] Canada, G. of, 2021, “Precise Point Positioning” [Online]. Available: https://webapp.geod.nrcan.gc.ca/geod/tools-outils/ppp.php?locale=en. [Accessed: 24-Dec-2021].

[39] Google, 2020, “Download Google Earth Pro for PC or Mac” [Online]. Available: https://www.google.com/earth/download/gep/agree.html?hl=en-GB. [Accessed: 28-Dec-2021].

[40] Takeshi, I., “Laptop Computer Isolated on a White Background with a Blank Screen” [Online]. Available: https://www.vecteezy.com/vector-art/376614-laptop-computer-isolated-on-a-white-background-with-a-blank-screen. [Accessed: 18-Jan-2021].

[41] ROS, 2021, “Ubuntu Installation of ROS Noetic” [Online]. Available: http://wiki.ros.org/noetic/Installation/Ubuntu. [Accessed: 05-Nov-2021].

[42] ROS, 2021, “ROS Concepts” [Online]. Available: http://wiki.ros.org/ROS/Concepts. [Accessed: 05-Nov-2021].

[43] MAVLINK, 2020, “Mission Protocol” [Online]. Available: https://mavlink.io/en/services/mission.html. [Accessed: 05-Jan-2021].

[44] Ermakov, V., 2021, “MAVROS” [Online]. Available: http://wiki.ros.org/mavros. [Accessed: 05-Jan-2021].

[45] Cadet, C., 2012, “Cub Cadet RZT-S Zero Professional Shop Manual” [Online]. Available: https://www.manualslib.com/manual/1065738/Cub-Cadet-Rzt-S-Zero.html. [Accessed: 26-Feb-2016].

[46] Inc., A. S., “BAC1000 Product Brochure” [Online]. Available:

https://www.tecknowledgey.com/amfilerating/file/download/file_id/569/. [Accessed: 26-Feb-2016].

[47] Kunze, M., and Accelerated Systems Inc., 2018, “ASI Modbus Protocol,” (Personal Email Communication w/ H.J. Sommer).

[48] Kunze, M., and Accelerated Systems Inc., 2018, “ASI Object Dictionary,” (Personal Email Communication w/ H.J. Sommer).

ACADEMIC VITA OF MICHAEL PAGAN

EDUCATION

The Pennsylvania State University, Schreyer Honors College • *Class of 2021* **University Park, PA**
College of Engineering • Bachelor of Science, Mechanical Engineering
College of Engineering • Minor, Engineering Leadership Development

TECHNICAL EXPERIENCE

Mechatronics Engineering Intern **Jun 2020-Aug 2020**
JLG Industries *Hagerstown Maryland*

- Independently managed a complex trade study project and successfully delivered results
- Utilized MATLAB to model and analyze steering cylinder forces throughout a static steer
- Acquired fluency with hydraulic schematics and technical drawings through critical review
- Culminated technical findings into Decision Analysis and Resolution to produce design solutions

Undergraduate Thesis Author **Sept 2019-Apr 2021**
Penn State Department of Mechanical Engineering *State College, PA*

- Developing autonomous all-electric zero-turn mower for an apple orchard

R&D Engineering Intern **Feb 2019-Dec 2020**
Penn State Applied Research Lab *State College, PA*

- Analyzed and repaired inoperable electronic lab equipment
- Collected, collated, and reported data on furnace temperature profiles
- Worked extensively with electric controllers and vacuum systems for furnaces
- Designed and built high temperature wet oxidation system
- Applied fundamental automation skills to furnace systems via PID control

LEADERSHIP EXPERIENCE

President **May 2020-May 2021**
Penn State Men's Club Volleyball University Park, PA

- Leads the 28-player club by heading all administrative relations and coaching
- Commits 10+ hours per week for club duties
- Supervises club team members and other executive officers
- Served as Vice President for the preceding year coordinating travel and assisting the President

INTERNATIONAL EXPERIENCE

Engineering Design Student **May 2018-June 2018**
Tecnun Universidad de Navarra *San Sebastián, Spain*

- Studied identification and resolution of global/cross cultural engineering problems
- Led design team that aimed to improve urban cycling in San Sebastián, Spain

ACTIVITIES AND HONORS

The President Sparks Award (4.0 GPA) *Jan 2019*
Louis A. Harding Memorial Scholarship (4.0 GPA) *Jan 2019*
The President's Freshman Award (4.0 GPA) *Jan 2018*
Penn State Dean's List *Dec 2017-present*
Penn State Men's Club Volleyball Team *Aug 2017-present*

SKILLS

Tech: SolidWorks, Creo, MATLAB, EES, Linux, RTK GPS, Python, Arduino, ROS
Engineering: Electrical troubleshooting; Mechatronics; Data collection and analysis; Schematic fluency; Hands-on fabrication