

Control Theory Implementation on RC Hovercraft

by:

Joshua Bevan

Nicholas Lutz

Ryan Mackay

Mario Stamatiou

Capstone Project for Spring 2013

Department of Mechanical Engineering

Francis College of Engineering

University of Massachusetts Lowell

Signature of Authors:

Capstone Advisor: Ioannis Raptis

Professor, Department of Mechanical Engineering

# Abstract

A hovercraft platform was constructed from a modified kit. A microprocessor with integrated IMU was used in conjunction with serial radio modules for teleoperation. A theoretical model was developed examining forces and moments acting on the platform. This model was then linearized and used as a basis to develop a control algorithm. The control algorithm allows input of waypoints and using data from the IMU autonomously navigates from point to point. A first pass at optimization of PID parameters was also performed.

Several tests were performed to examine the accuracy of the platform in following the waypoints, control stability, and robust error handling in response to unknown external perturbations. In all tests the hovercraft performed satisfactorily and results are presented showing system behavior.

Finally an introductory guide to construction, operation, and hardware and software modification are presented in several appendices to allow future development by new experimenters looking to expand the platform’s capabilities and performance

# Acknowledgements

We would like to acknowledge the efforts of Professor Raptis in acting as our capstone advisor. His contributions to our understanding of the theoretical and practical implementations of the control algorithm were invaluable. We would like to thank all the professors of the Mechanical Engineering Department for providing us the knowledge that was applied in successfully achieving the goal of this project. Additionally, we would like to thank RC Buyer’s Warehouse of Nashua, NH for providing advice on equipment selection.

# Table of Contents

[Abstract 2](#_Toc355060684)

[Acknowledgements 3](#_Toc355060685)

[Table of Contents 4](#_Toc355060686)

[List of Figures 6](#_Toc355060687)

[List of Tables 7](#_Toc355060688)

[Introduction 8](#_Toc355060689)

[Literature Review 9](#_Toc355060690)

[Project Schedule 10](#_Toc355060691)

[Theory 11](#_Toc355060692)

[The moment generated from the thrust force in the sway-direction will be 15](#_Toc355060693)

[Design Methodology 19](#_Toc355060694)

[Discussion of Procedures and Methods Used For Design 20](#_Toc355060695)

[Discussion of Methods Used for Analysis or Test, as well equipment used to test the design 23](#_Toc355060696)

[Results Obtained and Analysis Performed 24](#_Toc355060697)

[Efficiency of Results 29](#_Toc355060698)

[Summary and Conclusions 30](#_Toc355060699)

[Recommendations For Further Study 31](#_Toc355060700)

[Project Postmortem 32](#_Toc355060701)

[Teamwork 32](#_Toc355060702)

[Technical Communication Skills 32](#_Toc355060703)

[Project Schedule 32](#_Toc355060704)

[Ethical Standards 32](#_Toc355060705)

[Industrial Commercial Standards 32](#_Toc355060706)

[Safety 32](#_Toc355060707)

[Environmental Impact 32](#_Toc355060708)

[Societal Impact 33](#_Toc355060709)

[Multi-Disciplinary Issues 33](#_Toc355060710)

[Professional Societies Codes and Standards 33](#_Toc355060711)

[Bibliography 34](#_Toc355060712)

[Appendix A: Part List 35](#_Toc355060713)

[Miscellaneous Parts 38](#_Toc355060714)

[Appendix B: Construction Details 39](#_Toc355060715)

[Appendix C: Wiring Guide 41](#_Toc355060716)

[Appendix D: Wiring Diagram 42](#_Toc355060717)

[Appendix E: ArduRover Function List 43](#_Toc355060718)

# List of Figures

[Figure 1: Schedule of project proceedings 10](#_Toc355060719)

[Figure 2: Mission Planner Overview 11](#_Toc355060720)

[Figure 3: Inertial frame and body frame of hovercraft 12](#_Toc355060721)

[Figure 4: Rudder displacement 14](#_Toc355060722)

[Figure 5: Cross-section of thrust fan with rudder displaced 14](#_Toc355060723)

[Figure 6: Hovercraft configuration and forces applied 15](#_Toc355060724)

[Figure 7: Overview of real-time program flow 20](file:///C:\Users\Root\Desktop\Draft2.docx#_Toc355060725)

[Figure 8: Various testing waypoint setups 23](#_Toc355060726)

[Figure 9: Un-optimized track test 25](#_Toc355060727)

[Figure 10: Crosstrack error optimized track test 26](#_Toc355060728)

[Figure 11: Steering/crosstrack optimized along with box test results (box waypoints hidden for clarity) 27](#_Toc355060729)

[Figure 12: Testing of system stability dependent on initial conditions 28](#_Toc355060730)

[Figure 13. Location of Lift-Fan Hole 39](#_Toc355060731)

# List of Tables

[Table 1. Parts for Hovercraft Body 35](#_Toc355060732)

[Table 2: Assorted additional parts 38](#_Toc355060733)

[Table 3. APM Input Wiring Guide 41](#_Toc355060734)

[Table 4. APM Output Wiring Guide 41](#_Toc355060735)

# Introduction

Hovercrafts present unique control challenges; they are non-linear under-actuated systems. As a result they are difficult to operate manually. The desire of the capstone is to implement a linearized control algorithm using inertial data and a GPS on a microprocessor. A real-time program loop autonomously controls the hovercraft and allows it to navigate between preset waypoints.

Platform construction details are presented, and results will be presented demonstrating robust control of the platform.

# Literature Review

22.451 Dynamic Systems Analysis: The dynamic behavior and the control theory applied for the control algorithm were taught by Professor Raptis of the Mechanical Engineering Department

22.581 Advanced Fluid Mechanics: The fluid analysis used on the hovercraft was studied in the course taught by Professor Willis of the Mechanical Engineering Department

22.213 Dynamics: The kinematics and dynamics of the hovercraft were based on concepts in the course, taught by Professor McKelliget of the Mechanical Engineering Department

“Arduino Robotics”, Warren, John-David: The contents in this book on PID parameters provided us with an insight as to how to develop the control algorithm

“Hovercraft PID Control”, Raptis Ioannis: The concept for an equivalent application of our hovercraft was documented in this PowerPoint presentation

# Project Schedule



Figure 1: Schedule of project proceedings

# Theory

The main objective of the hovercraft was to autonomously travel from one point to another. The set-points were in the form of GPS coordinates provided through Mission Planner which would communicate with the hovercraft via MAVLink. When the control algorithm is developed, the PID parameters will be adjusted through Mission Planner to regulate the performance of the hovercraft.

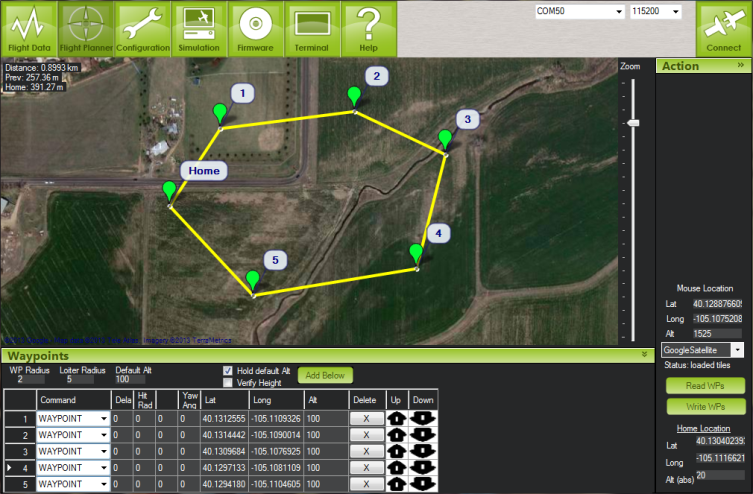


Figure 2: Mission Planner Overview

As seen in Figure 1, Mission Planner is in the form of a GUI in which the user can manually input the points of interest. The concept which was applied in the autopilot, so the hovercraft was able to identify where the points of interest are located, was line of sight. Line of sight is simply the ability to identify objects along an imaginary line between the observer, and the object which is being identified.

The motion of the hovercraft requires two stages, turning and cruising. In the first stage, if the hovercraft starts at an initial angle with respect to the line of sight or the imaginary line, then it would have to turn until it is in line with the point of interest. Once the first stage is achieved, the second stage would follow, which is when the hovercraft would translate until it is within a specific distance from the point interest. The control algorithm would reduce the amount of error when the hovercraft is turning, as well as when it is translating closer to the point of interest. In order to develop an effective control algorithm, the kinematics and dynamics of the hovercraft had to be understood.

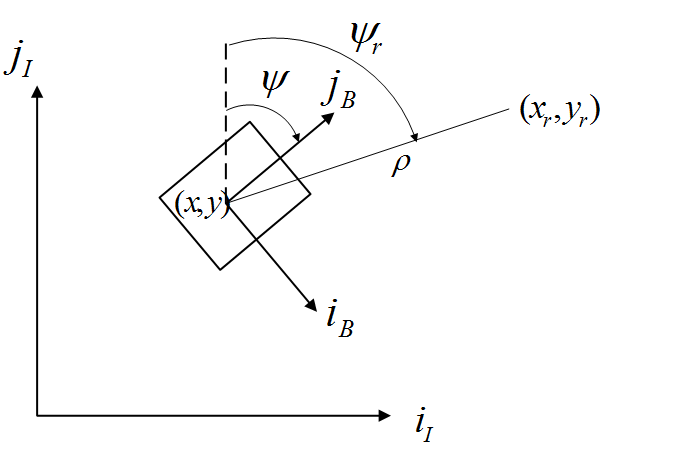


Figure 3: Inertial frame and body frame of hovercraft

As seen in Figure 2, the analysis of the motion of the hovercraft involved using the inertial frame and the body frame. The inertial frame refers to the Earth being the point of reference, so for any displacement, velocity, or acceleration being measured, the initial corresponding value is that of the Earth.

The dynamics of the Earth are only considered when dealing with high-speed applications such as supersonic jets or long-range missiles. For the hovercraft, it is assumed that the motion of the Earth does not affect the relative velocity or acceleration, so the inertial frame is considered to be at rest. The body frame refers to the hovercraft being the point of reference.

The components ib and jb are referred to as sway and surge respectively. The angle ψ is instantaneous angle with reference to the inertial frame, and ψr is the set-point angle in the inertial frame. The point (xr,yr) is the set-point and (x,y) is the point at which the hovercraft is located at a given instant with respect to the inertial frame.

The initial angle ψr is the angle at which the hovercraft must turn in order to be in line of sight with the point (xr,yr). At each instant when the hovercraft is turning, the angle ψr is always changing. Let the difference between the reference angle and the instantaneous angle be

(1)

where

ψr= (2)

When the hovercraft aligns with the set-point, then . In reality this is very difficult to achieve, so an error angle ε is defined. Once the hovercraft reaches within the vicinity of this angle, it is considered to be in line of sight. The condition by which this happens is

(3)

As the hovercraft approaches the set-point, the distance ρ decreases. From the two defined points, ρ can be obtained by applying the Pythagorean Theorem

(4)

A waypoint radius R is defined in the event that the hovercraft overshoots its set-point. The hovercraft will have reached its destination when or when it is within the waypoint radius In that case, the whole process of turning and cruising will repeat. The condition by which the hovercraft will have successfully reached the target point is

(5)

Unlike a plane or a quad copter which requires the knowledge of pitch and roll angles, the hovercraft does not translate in the z-direction or rotate about the surge axis or sway axis, so the motion is 2-dimensional, which was the reason why APMRover v.2 was selected as the firmware for the ArduMega autopilot, as to simulate the motion of a rover rather than that of an aircraft. Thus, there was no need to account for forces in the z-direction or moments occurring about the surge-axis or the sway-axis.

The two parameters of interest were yaw and thrust, where yaw is the angle of the hovercraft about the z-axis, and thrust is the force exerted from the thrust fan which acts in the surge-direction on the hovercraft. Due to the low viscosity of the air, and because the hovercraft will be going at a relatively high velocity, there is no need to account for Stoke’s flow which would result in an increase in viscous forces. Also, the hovercraft is to be tested on a relatively smooth surface, so the coefficient of kinetic friction will be very low. As a result, frictional forces and drag forces on the skirt are considered negligible. An important note is that the lift fan is decoupled from the thrust fan and the rudder, so any amount of lift generated does not affect the amount of thrust and yaw angle generated. The lift was regulated manually through the radio transmitter.

In fluid flow, when the path of a streamline is altered, the streamline follows that path. In this case, when the rudders of the hovercraft are displaced, the flow of the thrust fan is re-directed such that it is flowing tangent to the surface of the rudder. When this happens, the direction of the flow changes, and from just having a surge component, the velocity of the flow has a surge and sway component. From the Conservation of Linear Momentum for fluid flow, when there is a change in the direction of the flow, a force is generated. That force will have a surge and sway component.

The sway component is responsible for generating a moment, which causes the hovercraft to turn. While the rudders are at an angle, a portion of the cross-section of the fan will have air going in the surge-direction. The following concepts apply for one rudder as well, since the two rudders on the hovercraft being used both exert the same amount of forces in the surge and sway components.

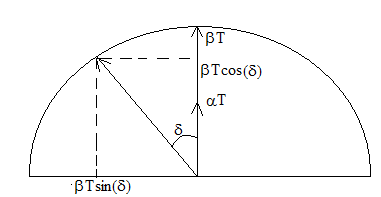


Figure 4: Rudder displacement

Figure 3 depicts a representation of the hovercraft for one rudder at an angular position δ, as to obtain a relationship of how much thrust force is generated vs. the angle at which the rudder is displaced

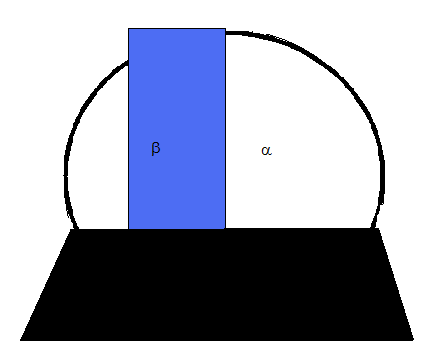


Figure 5: Cross-section of thrust fan with rudder displaced

The structure of the shroud which supports the thrust fan is shown in figure 4. The blue colored portion is the rudder and the white portion is the cross-section of the thrust fan in which there is no flow re-directed.

Let α be the percentage of the cross-section in which there is flow in the surge-direction which is the white part in figure 3, and β be the percentage of cross-sectional area that is covered by the rudder which is the blue portion. The total percentage of thrust is then

α+β=1 (6)

meaning that all the thrust that is generated by the fan is goes through the rudders, regardless of which direction.

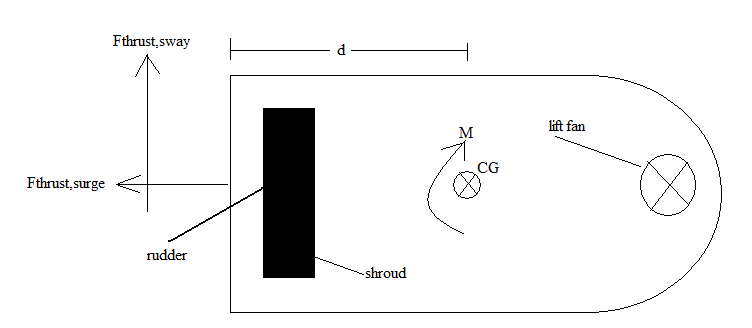


Figure 6: Hovercraft configuration and forces applied

If the rudder is at an angle δ, then the surge-component of the thrust force will be

Fthrust,surge= T(α+βcos(δ)) (7)

and the sway-component of the thrust force will be

Fthrust,sway=βTsin(δ) (8)

The moment generated from the thrust force in the sway-direction will be

M= dFthrust,sway=dβTsin(δ) (9)

where d is the distance from the center of gravity of the hovercraft to where the rudders attach to the shroud of the thrust fan.

The relations describing the forces and moment of the hovercraft are not linearly proportional to δ, so the system is non-linear. For the control algorithm, it is much easier if the system is linear, so control theory can be applied. The small angle approximation is used to linearize (7),(8) and (9). Thus, the previous relations can be simplified to

Fthrust,surge=T (10)

Fthrust,sway=βTδ (11)

M=dβTδ (12)

Having completed the kinematic analysis, the dynamic analysis can be implemented. Interpreting the sway and surge in terms of the velocities for the inertial frame, the following equations are obtained where *u* is surge and *v* is sway.

(13)

(14)

(15)

where and are the translational velocities for the ji and ii axes respectively, and is the angular velocity of the hovercraft.

When (13), (14) and (15) are differentiated, the accelerations are obtained. With reference to the inertial frame, the acceleration can be expressed as follows:

(16)

(17)

(18)

From (16), the coefficient of the cosine corresponds to the surge component of the acceleration in the body-frame. Similarly, the cosine term in (17) corresponds to the acceleration in the sway direction.

Applying Newton’s 2nd Law in the body frame, the corresponding acceleration for each force is obtained. As mentioned earlier, any frictional forces are disregarded, so the equations can be simplified to:

(19)

(20)

(21)

Having achieved a relationship between the accelerations and the forces and moments in the body-frame, the control algorithm can then be developed. By being able to control the amount of thrust produced, in turn, the acceleration of the hovercraft can be controlled as well, thus reducing overshoot when approaching the destination point. Controlling the thrust force will also have an effect on the moment applied to the hovercraft, which will have an effect on turning.

The previous equations were linearized, as to apply basic concepts of control theory. Since the variables will become deviation variables, a reference point has to be defined as to add each deviation point. The reference points are so , and As mentioned, any frictional forces are disregarded, so damping constants b and bω can be ignored, and because the initial point of reference is at rest, the deviation variables can be treated as state variables, so taking the derivative of (19), (20) and (21) and simplifying,

(22)

=> (23)

where c1 and c2 are constants obtained experimentally when testing the hovercraft. The control algorithm will focus on steering and cruising, so the equations of importance are those of surge acceleration and angular acceleration. Equations (22) and (23) relate the surge acceleration with the percentage of thrust, and the angular acceleration with the angular deflection of the rudder.

Control in a dynamic system is achieved when the amount of input to the system can be controlled, based on the desired output. This will be achieved by using the controller in the autopilot. In (22) and (23), the inputs of the system are the amount of thrust, and the angular deflection of the rudder. A PID controller is type of controller used in the control algorithm. The values of Kp KI and KD are trivial, so experimentation is required in order to acquire those values, to achieve a desired output.

For the hovercraft, the following PID loops will be applied:

For turning:

T%= (24)

∆%=-Kψsineψ-Krr (25)

where eψ is the error angle of the hovercraft with the angle of the set-point. For (27), a default value of thrust is applied when the hovercraft is turning, as to avoid sway velocities, which can throw off the hovercraft from its path.

For cruising, there are two scenarios. The first scenario consists of a PID loop similar to that of turning

T%=Kρρ-Kuu (26)

∆%=-Kψ’sineψ-Kr’r (27)

In (29) the thrust depends on the distance of the hovercraft from the destination point, so the greater the distance ρ, the more thrust that needs to be applied in order to get to that point faster. While cruising, the hovercraft has to be limited to the amount of moment applied, so Kψ and Kr’ are relatively small values, as to avoid great amount of angular displacement of the hovercraft.

The second scenario involves the hovercraft having low amounts of thrust. The slower the hovercraft travels, the less moment required, thus easier to turn. The PID loop for the thrust remains the same. For the angular displacement of the rudder

∆%=-sineψ-r (28)

where Kψ’’ and Kr’’ can be larger values, since less thrust means the hovercraft can turn more easily. With this scenario, if the thrust is too low, then ∆% increases dramatically which can affect the system, so a there has to be a saturated value of so that under that amount, the hovercraft will turn at that default value.

# Design Methodology

When selecting a hovercraft design to pursue, the first decision to be made was whether to use a ruddered vehicle or one with differential thrust. A ruddered hovercraft uses a single thrust fan with a set of rudders at the exhaust for redirecting flow. Differential thrust refers to a vehicle with two separate and symmetrical thrust fans. Rather than using a rudder, a turning moment may be achieved by passing a different amount of air through each of the propellers, or potentially running one in reverse. The result is that differential thrust allows the craft to turn in place by causing each motor to produce equal thrust with one in full reverse. Conversely, in order for a ruddered vehicle to turn, it requires air flow past the rudders. This means that whenever the vehicle needs to turn, it requires forward thrust, thus eliminating the possibility of turning in place. This was seen to be an interesting control problem and consistent with the spirit of the capstone, so the team decided on a single ruddered thrust fan for the final design.

The next design decision to be made was which mechanism to use for lift. The two dominant practices in model hovercrafts are to use either a duct or a separate lift fan. The first design uses a duct to redirect some of the air flow through the thrust fan to the skirt as the source of overpressure. Alternatively, a separate lift fan could be mounted to pass air directly to the skirt. The duct design requires forward thrust in order to be hovering which directly couples the thrust force to hover height. This causes a number of problems such as the necessity to be traveling at a minimum speed to hover and changing the speed also changes the traction. The necessity of a minimum speed was undesirable for the control algorithm, so a separate lift fan was implemented.

The final major choice to be made concerning the hovercraft design was which microcontroller to use. Professor Raptis helped to narrow the choice to the PX4 and ArduPilot Mega 2.5. The PX4 is an ARM processor developed by ETH Zurich and boasts a more powerful processor, but for our purposes, the higher processing speeds offered by the PX4 are unnecessary. The two boards have comparable sensor suites and both interface with MAVLink, although the APM does so more smoothly. The APM is arduino compatible, has much more clearly commented firmware and a greater amount of online support, making it preferable to the PX4.

# Discussion of Procedures and Methods Used For Design

Knowledge of how information flows through our system is integral to the implementation of the control algorithm. Figure 7 shows a block diagram of the MAVLink-ArduRover system.

The flow of information starts with the user in Mission Management who inputs parameters and waypoints to MAVLink. There is then a one-time request of parameters to the APM and then it begins autonomous navigation. The sensor values from the IMU are passed to programs that convert the readouts to coherent values and units. These values, such as position, attitude and speed, are then passed to the control algorithm.

Figure : Overview of real-time program flow

MAVLink

Mission Management

RC Input

Actuator Interface

Mixer

Position Estimation

Speed Control

Attitude Control

Attitude Estimation

Sensor Readout

Ang. Rate Control

Speed Estimation

Control Algorithm

Mission Logs

It is the job of our control algorithm to take in inertial measurements and determine what the actuator output should be. This translated to a PWM signal and passed along to the ESC’s which directly control the actuators (servo and thrust motor). The lift fan is controlled directly by the RC controller as a failsafe. The actuator activity causes a physical result to the hovercraft which influences the state of the inertial sensors, starting the information flow process again. At this point, the APM also updates the ground control station with the sensor output which is logged for later use.

The control algorithm outlined in (24), (26) and (28) needed to be transcribed to C++ to be implemented in the ArduRover software. The code needs to be split in to two cases defined by the bearing error () organized by a switch or if/else statement. The variables for ground speed, distance to waypoint, angular velocity, and bearing error are all accessible within the code for use. However, due to the structure of the code, it is more practical to control target speed than desired thrust. For similar reasons, it made sense to scale ∆% by ground speed rather than T% in (28). For easy access to Kρ,Ku, Kψ, and Kr during tuning, it made sense to use the existing PID class. The pseudo code to be implemented is as follows:

1|PID ρ\_pid, u\_pid, Ψ\_pid, r\_pid;

2|if ( |bearing\_error| < max angle for cruise )

3| Target\_speed = cruise\_speed + ρ\_pid( distance\_to\_waypoint, kp=Kρ , ki=0, kd=0 )

4| Target\_speed = Target\_speed + ρ\_pid( ground\_speed, kp=Ku , ki=0, kd=0 )

5|else

4| Target\_speed = cruise\_speed

5|T% = calc\_throttle( Target speed )

6|Limit T%min ≤ T% ≤ T%max

7|∆% = Ψ\_pid( sin(bearing\_error), kp=Kψ , ki=0, kd=0 )

8|∆% = r\_pid( omega.z, kp=Kr , ki=0, kd=0 )

9|∆% = (∆%)(cruise\_speed/ground\_speed)

The PID class contains a function that performs a standard PID loop on the argument; for example:

Where kp, ki, and kd are passed separately during tuning. Because and, they cannot use a single PID loop. To get around this, each equation uses the sum of two PID functions that only use the proportional terms with.

The hovercraft turning or cruising mode selection is controlled by the if/else statement in line 2 above. If the bearing error is smaller than a tolerance, lines 3 and 4 are executed, applying the velocity analog of (26). Otherwise, it skips to line 4, implementing (24). In either case, line 6 saturates the value of T% to not fall outside an allowed window. Lines 7 and 8 apply the control for rudder deflection given by (27). Line 9 reduces the gain for high ground speeds, effectively adapting the algorithm to (28).

# Discussion of Methods Used for Analysis or Test, as well equipment used to test the design

Multiple components of the project needed to be evaluated for performance and efficiency. Individual subsystems were tested for performance including the battery system as well as the ESC/motor systems for the thrust and lift fans. A Turnigy MegaMeter 7in1 Multimeter was used to evaluate motor current draw and battery performance. The batteries and motors were tested under typical and extreme discharge situations to estimate average and worst-case scenario performance.

Software level performance consisted of testing on the PID loops and control algorithm. The optimality of tuning for the PID parameters was evaluated in straight-line, box, and track type testing courses. The telemetry data for the parameter of interest was compared to the desired set-point and changes to the tuning were qualitatively judged in an attempt to minimize overshoot, and response lag. The control algorithm was tested under “real world” conditions in a track type test. Additionally the stability of the ruddered component of the algorithm was tested using a 3 point initial conditions test. Examples of these test types can be found in Figure 8 below.

Telemetry data was analyzed post-run in Mission Planner from the logs generated. Especially in the case of track type testing a “holistic” approach was adopted in balancing the myriad of factors that could be adjusted in relation to externalities beyond operator control. The ultimate goal of this analysis was to tune the system to be as fault-tolerant and robust as possible while maintain controllability, speed, efficiency, and ease of use.

Figure 8: Various testing waypoint setups

# Results Obtained and Analysis Performed

The resulting code used in the ArduRover firmware is given below.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Calculate target speed -LUTZ //

//

static void calc\_speed\_auto(void) //

{ //

static float VELOCITY = g\_gps->ground\_speed \* 0.01; //

float RHO = get\_distance(&current\_loc, &next\_WP) //

AP\_Float Speed\_calc = g.speed\_cruise; //

static int Theta\_MAX = 2500 //Bearing error switch for steering and cruising//

//

switch (control\_mode) //

case AUTO: //

case RTL: //

case GUIDED: //

if ( abs((int)bearing\_error\_cd) >= Theta\_MAX ){ //

g.speed\_auto.set( g.speed\_cruise ); //

} else { //

Speed\_calc += g.pidAutoSpeed\_p.get\_pid( RHO ); //

Speed\_calc += g.pidAutoSpeed\_d.get\_pid( VELOCITY ); //

} //

break; //

case STEERING: //

case LEARNING: //

case MANUAL: //

g.speed\_auto.set( g.speed\_cruise ); //

break; //

case HOLD: //

case INITIALISING: //

break; //

} //

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Calculate the required turn of the wheels rover -LUTZ //

//

static void calc\_nav\_steer() //

{ //

//

Vector3f OMEGA = ahrs.get\_gyro(); //Retrieve angular velocity –LUTZ //

//

// Adjust gain based on ground speed //

if (ground\_speed < 0.01) { //

nav\_gain\_scaler = 1.4f; //

} else { //

nav\_gain\_scaler = g.speed\_cruise / ground\_speed; //

} //

nav\_gain\_scaler = constrain(nav\_gain\_scaler, 0.2f, 1.4f); //

//

// negative error = left turn //

// positive error = right turn //

nav\_steer = g.pidNavSteer.get\_pid\_4500(bearing\_error\_cd, nav\_gain\_scaler); //

//

//Subtract a scaling term to penalize high turn rates -Lutz //

nav\_steer -= g.pidNavSteer\_d.get\_pid( (float)OMEGA.z) //

//

g.channel\_steer.servo\_out = nav\_steer; //

} //

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

The implementation of the control algorithm in C++ is almost precisely the same as the pseudo code explained earlier. The only major difference is the syntax used to interface with the rest of the ArduRover code. Also, the decision was made to pass Ψ rather than in the ∆% control. It was determined that the discontinuity experienced when was insignificant given the physics of the craft and its tendency to drift through it.

Once all subsystems were performing properly track type tests were done to evaluate “real-world” performance of the system. Waypoints were set on a satellite overlay of UMass Lowell’s track and the hovercraft was allowed to operate in “Auto” mode, autonomously navigating sequentially between waypoints. The circles around each waypoint represent the target area the hovercraft must reach before moving on. Figure 9 presents results before optimization was attempted.  


Figure 9: Un-optimized track test

Path adherence was found lacking in Figure 9. The crosstrack error gain was increased to more heavily enforce conformation to inter-waypoint vectors. The results are presented in Figure 10. Crosstrack error gain in this figure is set to 200 cm° (which relates the perpendicular pertubation to the corrective upstream angle).



Figure 10: Crosstrack error optimized track test

The more demanding line-following necessitated followup changes ot the PID paramenters to the steering loop to damp out the oscillations. Figure 11 presents the results of both optimizations in conjunction. Excellent adherence to the waypoints can be observed with deviation being no larger than ~1m



Figure 11: Steering/crosstrack optimized along with box test results (box waypoints hidden for clarity)

Figure 11 also presents the box-type test which more fully tests the control algorithms performance for more oblique path vertex intersections. Some overshoot on one of the waypoints is evident. In this case a choice must be made regarding desired system properties. If some overshoot is acceptable than the craft can proceed much more liberally. If path accuracy is important then aggressive tuning must be performed on both throttle and steering; this limits the generality of system capabilities.

Finally testing of the stability of the system was performed using a 3 point initial conditions test. This was performed twice to examine repeatability. Figure 12 presents the results. In all three test cases the hovercraft successfully navigated to the central goal with minimal turning overshoot and no control discontinuities evident.



Figure 12: Testing of system stability dependent on initial conditions

# Efficacy of Results

The results of the final testing regimens match the expected results. The system behaves in a relatively robust manner. As expected the linearization of the non-linear system has some shortcomings in situations where the small angle approximation used for rudder deflection no longer holds valid. This is important in tight turns, failure recovery, or in situations with large crosswinds that requiring tacking or sustained upwind crosstracking.

It is important to note the dependence of the quality of control on quality sensor input. In situations whre poor GPS conditions exist or where the update rate suffers, the control algorithm suffers from lack of positional accuracy. A complimentary dead-reckoning method is needed in parallel to handle such contingencies.

# Summary and Conclusions

A hovercraft platform was constructed that allowed ruddered thrust and independent lift to be realized. A microprocessor with IMU and GPS was integrated that allowed measurement of system attitude and position. A control algorithm based on relevant theory was implemented on the microprocessor. In conjunction with serial radios for duplex communications a ground control station was used to pass waypoints to the system. The system was able to autonomously navigate between these waypoints. Numerous PID tuning sessions and modification of other operational parameters was necessary to realize expected “real-world” performance.

Testing was performed on the platform, and using the qualitative data gathered from the telemetry logs this system was optimized until it was able to navigate a 400m meter track with a path deviation of no greater than ~1m.

# Recommendations For Further Study

During the course of this project, several avenues for further research were discussed if time permitted. One such topic was the autonomous control of the lift fan. It should be possible to measure the effects of the current surface under the craft, i.e. the roughness and whether it is in contact with the skirt. It would be an interesting control challenge to control the lift fan to get the optimum thickness of air cushion under the craft in response.

A parallel research area that could be explored is the control of a craft using differential thrust. The kinematics of the system, and thus the control algorithm, would be completely different with the addition of a second thrust fan. This would also allow the hovercraft to turn in place, opening windows not available to the team while using a ruddered craft. By decoupling the turning moment and thrust force, path optimization could be explored in greater detail.

# Project Postmortem

## Teamwork

The team was formed in November 2012 under Professor Raptis and comprised of Ryan Mackay, Josh Bevan, Nick Lutz and Mario Stamatiou. A schedule was formed around all four group members schedule taking into account work/school and transportation. An effort was made to include all group members in all key decisions and project work.

## Technical Communication Skills

During the course of the project email/phone was the primary means of communication between the group/advisor. Every Monday the entire group met with Professor Raptis to discuss last week’s progress and the current weeks schedule as well as multiple work sessions throughout the week. All emails were answered quickly and professionally.

## Project Schedule

The capstone project schedule has changed significantly throughout its life time with changes in platforms and hardware. Weekly schedules and goals were planned every Monday night and broader goals were planned monthly. Overall the schedule met with our project goals with some things taking significantly longer (builds time) and others taking shorter than expected.

## Ethical Standards

All data used was gathered from credible sources and given credit in the bibliography. Any problems the group encountered were solved respectfully and professionally. Any data represented in the report/presentation has not been altered or distorted.

## Industrial Commercial Standards

The hovercraft was treated as a fully autonomous vehicle and followed the safety of operating procedures outlined in the ArduPilot code site.

## Safety

When dealing with RC vehicles and LiPo battery safety was the upmost concern. All safety guidelines outlined by ArduPilot as well as the safe storage/use of the LiPo battery outlined by the manufacture.

## Environmental Impact

Both the Lipo and NiMH batteries contain toxic chemicals that can pose an environmental threat. All batteries will be appropriately used and recycled. The hovercraft itself is run entirely on electrical circuits and has no emissions. When operating the hovercraft safety as well as noise was considered in choosing the time and location.

## Societal Impact

The hovercraft will be used as a research platform in the Autonomous Robotics Systems Laboratory (ARSL). It will be used as a landing platform for quad rotors in professor Raptis’s research on autonomous systems.

## Multi-Disciplinary Issues

The implementation of the code onto the PPM involved the use of C++ programing as well as knowledge of the existing system. Also knowledge of control systems not explicitly taught in the ME curriculum was needed to develop the control algorithm.

## Professional Societies Codes and Standards

The non-standard nature of the system made relevant standards difficult to determine. Best practices were adopted based on similar autonomous vehicles. These practices were unable to be determined if they were codified in a technical resource.

# Bibliography

# Appendix A: Part List

Aero Raver R/C model Hovercraft

Cost: $150

Weight: 1.3kg.   
Dimensions: 750 x 450 x 330mm

Features: Deck and Hull made from correx and polystyrene, single motor for thrust and hover, single bag skirt, most parts pre-trimmed, skids attached to save damage to hull, universal motor mount, twin rudders allows operation in confined spaces. Hovers up to 25mm high.

.



Table 1. Parts for Hovercraft Body

|  |  |  |  |
| --- | --- | --- | --- |
| Correx Deck | 5v Servo Motor | Bag Skirt | Guard |
| Exp. Poly Deck | Wood Varnish | Wooden Rudders | Skids |
| Exp. Poly Hull | Electrical Tape | Pivot Pin | Aero Racer Reg No. |
| Screen | Velcro Tape | 90 degree Bell Cranks x2 | ½ Bots and Washers x12 |
| Duct | Bolts for Motor | Connecting Wire | 5 ft of 1/64 Plywood |
| Motor Bracket | 1/16 Wire | No.6 Screws x 13 | Foam Board Adhesive Sealant |

Mechanical Systems

**Cost: $16**

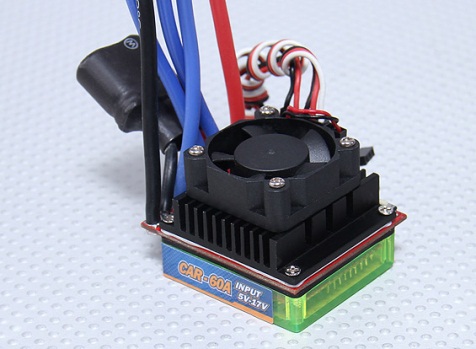
**Specs:**  
Model: **NTM Prop Drive Series 35-30 1100kv**  
Kv: **1100rpm/v**  
Max current: **32A**  
Max Power: 350W @ 12v (3S) / 380 @ 15v (4S)  
Shaft: **4mm**  
Weight: **88g**  
ESC: 40  
Cell count: **3s~4s Lipoly**  
Bolt holes: **18.9mm & 25mm**  
Bolt thread: **M3**  
Connection: **3.5mm Bullet-connector**

**For more information see http://www.hobbyking.com/hobbyking/store/\_\_16230\_\_NTM\_Prop\_Drive\_Series\_35\_30**

Thrust Motor



Speed Controller for Thrust Motor



**Cost: $26.99**

**Specification:**

**Model: Brushless Car ESC 60A w/ Reverse**Input voltage: 2-4S Lithium batteries / 4~15 Ni-xx  
Cont. Current: 60A  
BEC output: 3A/5V (Switch)  
Size( length X width X high): 47x41x29mm  
Weight: 82g

For more information see http://www.hobbyking.com/hobbyking/store/\_\_11743\_\_HobbyKing\_Brushless\_Car\_ESC\_60A\_w\_Reverse.html

Lift Fan

Cost: 17.99

Specification:

Model: EDF65 with 2730kv Motor Assembled

Blade Diameter: 65mm

Blade Type: 4

Shaft Size: 2.7mm

Voltage 11.1V

Max Current30amps

Max Thrust 600Grms thrust

For more information see :

http://www.hobbyking.com/hobbyking/store/\_\_8514\_\_EDF65\_with\_2730kv\_Motor\_Assembled.html.html



Lift Fan Speed Controller

**Cost: $10.75**

**Specfication.**Model:30A BlueSeries Brushless Speed Controller

Cont. Current: **30A**Burst Current: **40A**  
Battery: **2-4 Cell Lipo / 5-12 Cell Ni-XX**  
SBEC: **5V/ 3A Output**  
Size: **23\*43\*6mm**  
Weight: **28g**



APM Board

**APM 2.5+ Assembled (Top entry) with 915Mhz (US) Telemetry Set**

**Cost: $249.99**

**Features**:

* Arduino Compatible
* Includes 3-axis gyro, accelerometer and magnetometer, along with a high-performance barometer
* Onboard 4 MP Dataflash chip for automatic datalogging
* Digital compass powered by Honeywell's HMC5883L-TR chip, now included on the main board.
* Optional off-board GPS, Mediatek MT3329 or uBlox LEA-6H module (Mediatek included in base price; choose options at right subtract $20 for no GPS or add $50 for the better uBlox module).
* One of the first open source autopilot systems to use Invensense's 6 DoF Accelerometer/Gyro MPU-6000.
* Barometric pressure sensor upgraded to MS5611-01BA03, from Measurement Specialties.
* Atmel's ATMEGA2560 and ATMEGA32U-2 chips for processing and usb functions respectively.

For more information see:

http://store.diydrones.com/APM\_2\_5\_Assembled\_Top\_entry\_p/br-apmpwrkt-telem915.htm

[](http://store.diydrones.com/v/vspfiles/photos/BR-APMPWRKT-TELEM915-2.jpg)

GPS

**3DR GPS uBlox LEA-6**

**Cost: $75.99**

* ublox LEA-6H module
* 5 Hz update rate
* 25 x 25 x 4 mm ceramic patch antenna
* LNA and SAW filter
* Low noise 3.3V regulator
* I2C EEPROM for configuration storage
* Power and fix indicator LEDs
* Exposed RX, TX, 5V and GND pad
* 38 x 38 x 8.5 mm total size, 16.8 grams.

[](http://store.diydrones.com/v/vspfiles/photos/BR-3DRLEA-6-2.jpg)

**Total Cost of Main Components:** $ 498.62 (not including shipping costs)

## Miscellaneous Parts

Table 2: Assorted additional parts

|  |  |  |
| --- | --- | --- |
| Part | Purchase Location | Price |
| 22 Ga Solid Core Wire | Radio Shack | 8.49 |
| D-Sub Connector | Radio Shack | 2.69 |
| FTDI Serial Chip | Micro Center | 14.99 |
| Foam Adhesive | Home Depot | 3.68 |
| Sanding Sponge | HomeDepot | 2.97 |
| (5x) Balsa sheets 1/32x2x36" | R/C Buyers Warehouse | 6.36 |
| Sock cap screw 3mmx6 | R/C Buyers Warehouse | 1.51 |
| Sock cap screw 3mmx8 | R/C Buyers Warehouse | 1.51 |
| 6C 700 mAh NiCD | R/C Buyers Warehouse | 14.99 |
| Bullet Conn 3.5mm | R/C Buyers Warehouse | 2.24 |
| Prop Adapter 4mm | R/C Buyers Warehouse | 4.49 |
| (2x) EC3 Conn, Male | R/C Buyers Warehouse | 5.98 |
| (2x) EC3 Conn, Female w/4" wire | R/C Buyers Warehouse | 4.98 |
| 3200 mAh 4S LiPo | R/C Buyers Warehouse | 65.25 |
| Stain Pad | Belletetes | 1.29 |
| Wood Stain | Belletetes | 5.39 |
| Wood Glue | Belletetes | 3.59 |
| Mounting Tape | Home Depot | 12.31 |
| #6 Washer | Home Depot | 1.18 |
| #8 Washer | Home Depot | 1.18 |
| #6 Lock Nut | Home Depot | 1.18 |
| #8 Lock Nut | Home Depot | 1.18 |
| #6 Machine Screw | Home Depot | 1.18 |
| #8 Machine Screw | Home Depot | 1.18 |
| Velcro | Home Depot | 2.93 |
| APC 9x6 SF Prop | RC Buyers | 1.99 |
| (2x) EC3 Conn, Male | RC Buyers | 5.98 |
| (2x) EC3 Conn, Female | RC Buyers | 5.98 |
| NiMH 2000 mAh pack | RC Buyers | 20.99 |
| BEC Switch | RC Buyers | 5.95 |
| Electrical tape | Home Depot | 1.97 |
| Solder | Home Depot | 7.47 |
| 1.5" fitting | Home Depot | 0.97 |
| Pipe insulation | Home Depot | 2.36 |
|  | Total | $233.27 |

# Appendix B: Construction Details

The hovercraft construction was based on Aero Raver R/C model Hovercraft construction instructions (Patent No. 3010329-25 Jan03) with modifications. For the purpose of this report only the modifications will be described in detail for a more detailed construction guide see modelhovercraft.com.

**Main Structure Assembly**

The main structure was assembled as per the hovercraft instructions.

Parts Required- Correx Deck, Poly Deck, Poly Hull, Correx Hull, Sandpaper, Foam Board Adhesive Sealant,

1. Lightly sand non painted sides of both correx hull and deck. Making sure to keep Removable Correx pop outs in attached (will not be needed with modifications)
2. Assembly Order (starting at bottom) Correx Deck>Poly Deck,>Poly Hull> Correx Hull
3. Apply Pl 300 Foam Board Adhesive Sealant to mating sides of parts making sure to get an even coating of sealant.
4. Make sure to apple liberal amounts of sealant to any seam or airflow duct to prevent any pressure loss.
5. Add weights to strengthen seals and leave at least 24 hours to cure.

**Ply-Wood Reinforcements**

Ply-wood reinforcements were attached as per hovercraft instructions.

Parts Required- Hovercraft Body, 5ftx4in 1/64 ply, Wood Glue, Exacto Knife, Wood Varnish, Brush, Tape

1. Cut the ply to appropriate width (2 inches for hull, 4 inches for deck) around 5 feet for the deck and three feet for the hull will be needed(try to minimize the total pieces to avoid misalignments between pieces)
2. Attach the ply to the hull/deck edges using a liberal amount of wood glue making sure to tape the wood down as you attach it (Make sure to start and end at a straight part of the hull/edge to minimize stress on edges.) Allow appropriate amount of time to dry.
3. Once dry remove tape and apply varnish to wood. \*\*\*Test Varnish on spare foam!!!\*\*\*

**Installing Lift-Fan**

Parts Required-Lift-Fan, Drill, Knife, Guard

1. A 3 inch hole must be cut on the center axis of the bow of the hovercraft as seen below.

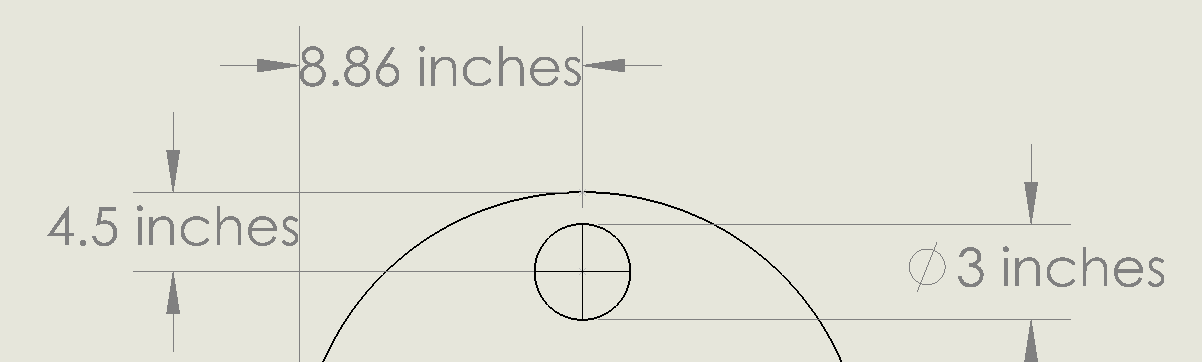


Figure 13. Location of Lift-Fan Hole

1. Coat the outer ribs of the fan with the Foam board sealent and put the fan into the cut hole making sure it has a snug fit (don’t jam it though!)
2. Seal both the outer and inner edges of the Fan.
3. Drill a hole through the hull bulkhead into the storage compartment.
4. Run the ESC wires through the bulkhead hole and seal hole on both sides.
5. Cut a large enough guard out of chicken wire to unwanted objects from getting sucked into lift fan such as fingers and secure it to hull using electrical tape.

**\*\*\*NOTE CHECK TO MAKE SURE ALL LIFT-FAN CONNECTIONS ARE WORKING AND SECURE BEFORE ATTACHING SKIRT!!!\*\*\***

**Attaching Skirt**

Skirt was attached as per hovercraft instructions.

Parts Required-Hovercraft Body, Skirt, Double Sided Tape, Electrical Tape

1. Apply tape along perimeter of hull/deck edge \*\*\*Do Not Remove Backing!!!\*\*\*
2. Dry assemble skirt on hull perimeter past the tape making sure to check for any creases or folds. Remove tape backing and slide skirt into position.
3. Firmly secure skirt with one electrical tape (two or three complete rotations around hull should be enough)
4. Repeat steps 2 for deck edge.
5. Firmly secure skirt to hull with electrical tape. 3 or more complete revolutions of hull edge.

\*\*\*Note if skirt separation occurs during flight re-apply electrical tape to hull\*\*\*

**Duct and Motor Assembly**

Parts Required-Duct Mount, Duct, Bolts/Washers, Motor Bracket, Propeller, Thrust Motor, Screws, Drill.

1. Cut out Duct Mount/Duct Assemble as per Hovercraft instructions.
2. Mount thrust motor to Motor Bracket using 25mm motor screws. Attach Prop to motor.
3. Align motor on duct mount making sure to center the motor as much as possible (making sure the prop is equidistant to all edges) and mark bolt positions.
4. Drill Bolt holes and bolt Motor Bracket to Duct Mount.
5. **\*\*\*DO NOT CUT OUT VENT HOLES\*\*\***
6. Cut Guard from chicken wire and attach with electrical tape.
7. Run motor lead wires around back of Duct.
8. Screw Duct Mount to Stern of Hovercraft (Centered about quarter inch from back edge) \*\*might want to wait on this step\*\*

**Rudder Assembly**

Rudder Assembly made as per Instructions.

Parts Required-Wooden Rudders, 1/8 piping, Servo, Velcro, Thin Gauge Wire, Bell Crank

1. Drill collinear holes in Duct/DuctMount for ruder pivots (Equidistant apart centered around motor)
2. Bolt wooden rudders to piping and attach to duct mount
3. To prevent the piping from falling through the Ductmount use something thread through the underside of the Duct Mount.
4. Glue Bell-Cranks to rudder rods and couple two with the thin gauge wire.
5. Attach Velcro to Servo and mount to side of duct mount.
6. Attach Bell Crank to Servo and couple with wire.
7. Run Servo lead around back of duct

**Wiring Guide.**

# Appendix C: Wiring Guide

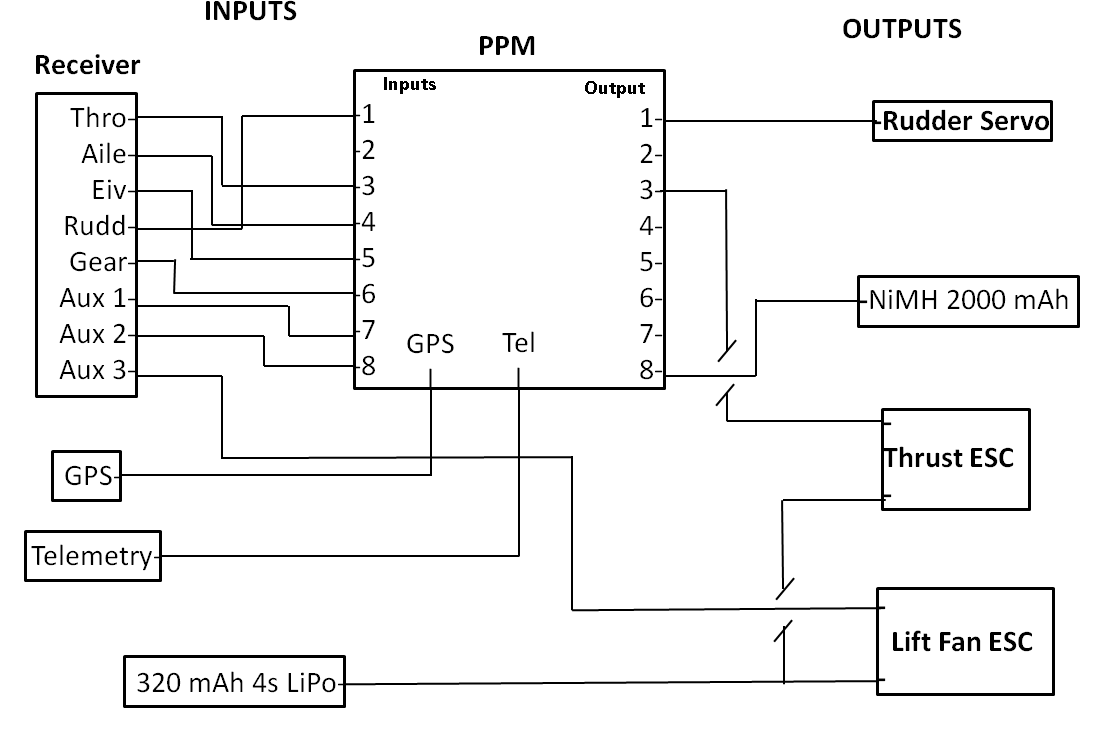
Table 3. APM Input Wiring Guide

|  |  |  |
| --- | --- | --- |
| **Receiver Output** | **Function** | **APM Inputs** |
| THRO | Thrust Fan | 3 |
| AILE | n/a | 4 |
| EIV | n/a | 5 |
| RUD | Rudder Servo | 1 |
| GEAR | n/a | 6 |
| AUX 1 | Momentary | 7 |
| AUX 2 | Modes | 8 |
| AUX 3 | Lift Fan | N/A (Goes directly to the Lift ESC bypassing board.) |
| **Additional Outputs** |  |  |
| U-Blox GPS | GPS | GPS |
| Telemetry | Telemetry | TEL |

Table 4. APM Output Wiring Guide

|  |  |  |
| --- | --- | --- |
| **APM OUTPUT** | **Function** |  |
| 1 | Rudder Servo | Rudder Servo |
| 3 | Thrust Fan | Thrust ESC |
| 8 | Board Power | NiMH 2000 mAh pack |

# Appendix D: Wiring Diagram



# Appendix E: ArduRover Function List

TYPE FUNCTION VARIABLE CLASS

**Function:** void setup()

**Description:**  Loads default values for variables; runs once on startup

**Definition in:** APMrover2.pde

**Functions Contained:**

**Used in: In Function:**

**Function:** void loop()

**Description:**  Main loop; runs continuously at ~50Hz, but synchronized with gyro/accel.

**Definition in:** APMrover2.pde

**Functions Contained:**

fast\_loop()

medium\_loop()

**Used in: In Function:**

**Function:** static void fast\_loop()

**Description:**  Loop running at 50Hz; outputs pwn signals and updates MAVLINK

**Definition in:** APMrover2.pde

**Functions Contained:**

read\_radio()

gcs\_update()

calc\_speed\_auto()

calc\_bearing\_error()

Log\_Write\_Attitude()

Log\_Write\_IMU()

update\_current\_mode()

set\_servos()

gcs\_update()

gcs\_data\_stream\_send()

**Used in: In Function:**

APMrover2.pde loop()

**Function:** static void medium\_loop()

**Description:**  Loop running at 10Hz; interfaces with GPS and performs navigation routines

**Definition in:** APMrover2.pde

**Functions Contained:**

failsafe\_trigger()

update\_GPS()

ahrs.set\_compass()

compass.null\_offsets()

navigate()

read\_receiver\_rssi()

update\_commands()

Log\_Write\_Control\_Tuning()

Log\_Write\_Nav\_Tuning()

Log\_Write\_GPS()

read\_trim\_switch()

slow\_loop()

**Used in: In Function:**

APMrover2.pde loop()

**Function:** static void slow\_loop()

**Description:**  Loop running at 3 1/3Hz; reads the mode set by the RC Transmitter

**Definition in:** APMrover2.pde

**Functions Contained:**

compass.save\_offsets()

read\_control\_switch()

update\_aux\_servo\_function()

update\_events()

**Used in: In Function:**

APMrover2.pde medium\_loop()

**Function:** static void calc\_speed\_auto(void)

**Description:**  Determines the target speed in auto mode

**Definition in:** APMrover2.pde

**Functions Contained:**

get\_distance()

set()

PID.get\_pid()

**Used in: In Function:**

APMrover2.pde fast\_loop()

**Function:** static void update\_GPS(void)

**Description:**  Reads the GPS and calculates current ground speed

**Definition in:** APMrover2.pde

**Functions Contained:**

ahrs.get\_position()

init\_home()

compass.set\_initial\_location()

**Used in: In Function:**

**Function:** static void update\_current\_mode(void)

**Description:**  Defines the behavior in each mode and executes auto calculations

**Definition in:** APMrover2.pde

**Functions Contained:**

calc\_nav\_steer()

calc\_throttle()

RC\_Channel. pwm\_to\_angle()

PID.reset\_I()

**Used in: In Function:**

APMrover2.pde fast\_loop()

**Function:** static void calc\_throttle(float target\_speed)

**Description:**  Calculates throttle values in auto mode

**Definition in:** Steering.pde

**Functions Contained:**

auto\_check\_trigger()

get()

get\_pid()

**Used in: In Function:**

APMrover2.pde update\_current\_mode()

**Function:** static void calc\_nav\_steer()

**Description:**  Calculates Rudder angles in auto mode

**Definition in:** Steering.pde

**Functions Contained:**

PID.get\_pid\_4500()

**Used in: In Function:**

APMrover2.pde update\_current\_mode()

update\_navigation()

**Function:** static void set\_servos(void)

**Description:**  Converts desired actuator settings to PWM and sends the signal

**Definition in:** Steering.pde

**Functions Contained:**

read()

RC\_Channel. calc\_pwm()

get()

throttle\_slew\_limit()

RC\_Channel. norm\_output()

write()

RC\_Channel\_aux. output\_ch()

**Used in: In Function:**

test.pde test\_radio()

APMrover2.pde fast\_loop()

**Function:** static void throttle\_slew\_limit(int16\_t last\_throttle)

**Description:**  Defines how quickly the Throttle value is allowed to change

**Definition in:** Steering.pde

**Functions Contained:**

**Used in: In Function:**

Steering.pde set\_servos()

**Function:** float PID::get\_pid(float error, float scaler)

**Description:**  Implements a generic PID loop

**Definition in:** PID.cpp

**Functions Contained:**

**Used in: In Function:**

Steering.pde calc\_throttle()

**Function:** int16\_t PID::get\_pid\_4500(float error, float scaler)

**Description:**  Returns the result of get\_pid() constrained by ±45°

**Definition in:** PID.cpp

**Functions Contained:**

PID.get\_pid()

**Used in: In Function:**

Seteering.pde calc\_nav\_steer()

**Function:** static void calc\_bearing\_error()

**Description:**  Calculates and filters the bearing error

**Definition in:** navigation.pde

**Functions Contained:**

wrap\_180\_cd()

butter10hz1\_6.filter()

**Used in: In Function:**

APMrover2.pde update\_navigation()

fast\_loop()

**Function:** static void navigate()

**Description:**  Calculates the desired heading as well as distance to next wp

**Definition in:** navigation.pde

**Functions Contained:**

get\_distance()

get\_bearing\_cd()

update\_navigation()

**Used in: In Function:**

APMrover2.pde medium\_loop()

**Function:**

int32\_t get\_bearing\_cd(const struct Location \*loc1, const struct Location \*loc2)

**Description:**  determines the bearing from loc1 to loc2 in the inertial frame

**Definition in:** location.cpp

**Functions Contained:**

**Used in: In Function:**

commands.pde set\_next\_WP()

navigation.pde navigate()

reset\_crosstrack()