April 2014

Sailboat Stabilization System

Dominic Gonzalez
Worcester Polytechnic Institute

Michael Eric Brendlinger
Worcester Polytechnic Institute

Pedro John Miguel
Worcester Polytechnic Institute

Steven Mize Rutledge
Worcester Polytechnic Institute

Sailboat Stabilization System

A Major Qualifying Project Report Submitted to the Faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science By

Michael Brendlinger

Dominic Gonzalez

Pedro Miguel

Steven Rutledge

April 2014

Approved by

Kenneth A. Stafford

William R. Michalson

Abstract

Sailboats have played an integral part in history and development of modern society. The scope of this project focuses on one particular aspect of the operation of this boat: the crew. The crew is the secondary boat operator sitting toward the front of the boat and the primary need for the crew is to simply shift his/her weight in the boat in order to keep the boat at the desired angle. By replicating this action with an autonomous device, the crew can be eliminated. By setting a desired heel angle on a control panel, a mass driven by a motor can be moved laterally along a track in order to adjust the heel angle accordingly. The goal of this project is to allow a sailor to safely operate a Flying Junior dinghy without a crew and be able to maintain full control of the boat.

Acknowledgements

Our team would like to thank the following for their assistance in the manufacturing process and their expertise in their respective fields:

Department of Electrical and Computer Engineering:

Professor William R. Michalson. MQP Advisor

Robert M. Boisse, Electronics Technician III

Department of Robotics Engineering:

Professor Kenneth A. Stafford, MQP Advisor

Joseph St. Germain, Robotics Lab Manager

Tracey Coetzee, Robotics Administrative Assistant

Kevin Arruda, Higgins Lab Manager

Authorship

Table of Contents

Table of Figures

Table of Tables

Executive Summary

Throughout history sailboats have been involved in the daily lives of human beings, built and controlled by manpower. The more recent sport of sailing and development in robotics presents an opportunity for change. This MQP team posed the question: what if a robotic system could replace the role of a sailor. The scope of this project focuses on one particular aspect of the operation of a boat: the crew. The crew is the secondary boat operator sitting toward the front of the boat and the primary need for the crew is to simply shift his/her weight in the boat in order to keep the boat at the desired angle of heel. What if this motion could be replicated with an autonomous device; could it replace the crew? In order to understand the details of this problem the team did an analysis of the role of the crew as well as the Flying Junior Dinghy in order to develop an understanding of the requirements for the robotic system.

The purpose of the project was to try to replicate the motion of a crew and in order to do that the team analyzed the crew's role in two ways. The first and simpler method was to take the average crew and determine how much righting moment they could generate. The selected crew size was a person 5 feet and 8 inches tall and weighing 150 pounds. They were able to generate 425 ft-lbs of righting for by hiking out the side of the Flying Junior. In order to further understand the true righting force needed by the system a full system analysis was done. By using the sail area of a Flying Junior while close-hauled, measured and calculated centroids of the sail and centerboard, and a maximum wind condition of 15 knots, the team determined that a device would need to generate 398 ft-lbs of torque in order to hold the boat at a constant heel. The first approach yielded a higher torque so the 425 ft-lbs was selected as our project's righting moment requirement. The second very important part of the analysis was

determining the location on the boat within which the system could operate. This was important because the deck of the Flying Junior vessel is very crowded so the system must be clear of the other parts of the boat, such as the mast and rigging, in order to not impact operation. Measurements were taken and the available space was noted and used as a parameter during the design phase.

Several challenges were present in designing a device to accomplish this task. First, all of the electronics involved would need to be sealed in waterproof boxes to prevent shorting due to splashing water. Second, all of the materials used need to be resistant to rust and corrosion, also due to the wet and potentially salty environment. Finally, energy restrictions were an issue when dealing with an isolated system that requires a significant amount of power. Batteries are heavy, and keeping the weight of the device less than the weight of an average crew is another important restriction. The final design of our project is able to overcome all of these challenges. Dry boxes were used to keep all electronics safe, 6061 aluminum alloy was used for all track components, and a 70 pound marine battery was selected that will provide 10 hours of continuous use on a single charge, and will also be used as the mass that creates the force necessary to stabilize the boat.

The electrical system consists of a custom designed PCB board containing a microprocessor, gyroscope, magnetometer, accelerometer, and inputs for the various sensors and user controls. This PCB, along with a Talon motor controller, is housed in a waterproof box. The microprocessor receives input signals that determine the position of the boom, position of the mass along the track, the heel angle of the boat, and the desired heel angle set by the skipper. With these inputs, the microprocessor is able to send PWM signals to the motor controller in order to adjust the position of the mass resulting in a righting moment that will heel the boat to within a threshold of the desired heel angle.

The design of the mechanical system is a horizontal two-stage bidirectional telescopic slide. This allows the track and mass system to be extended beyond the edge of the boat on both sides while heeled, and completely within the boat when centered. Movement of the mass along the track is controlled by a CIM motor driving a chain and sprocket assembly for the first stage and a rope and pulley system for the second stage. The two stages of the track are mounted with sealed ball bearings to reduce friction. An easy to remove clamp system was also implemented to make installing and uninstalling of the entire device quick and easy and without requiring any modification to the boat itself.

The final system successfully implemented the use of a Inertial measurement unit to determine the angle of heel and a PID control loop to maintain the desired heel angles. The system's final weight was 130 pounds, underneath the average crew weight of 150 pounds. It was able to generate a righting moment of 466 ft-lbs, higher than the 425 ft-lbs determined during our analysis. Functionality was tested in the laboratory on a custom built testing apparatus. The results showed that the system was in fact able to constantly adjust to hold the set heel angle and perform correctly based off of a series of user controls. The system performed the desired tasks stated in our project proposal: easy to install, weighs less than 150 pounds, provides at least 425 ft-lbs of righting torque, and can operate for several hours on one charge.

1.0 Introduction

Sailboats have played an integral part in the history and development of modern society. The oldest known record of a boat driven by sails dates back to 5000 BC. For thousands of years, sailboats were the optimal mode of transportation through water and were used for commercial transportation, fishing, and warfare. Today, sailboats are rarely used for commercial purposes; however, sailing has become a widely enjoyed sport and recreational activity. Sport sailing has been around since the 17th century and has since spread worldwide with hundreds of different classes and types of races.

This project focused primarily on boats involved in sport sailing, specifically dinghy vessels. The boat selected for this project is the Flying Junior dinghy, and sailing a boat this size requires two people to operate. However, there are many situations in which someone may either want to sail alone, or just doesn't have a sailing partner available. If one tries to sail alone, he/she is extremely susceptible to capsizing due to not having enough weight to counter the roll of the boat. These factors make sailing alone in a two person boat a very unwise choice.

The scope of this project focused on one particular aspect of the operation of this boat: the crew. The crew is the secondary boat operator sitting toward the front of the boat and the primary need for the crew is to simply shift his/her weight in the boat in order to keep the boat at the desired angle of heel. This action can be performed both autonomously and more efficiently. By replicating this action with an autonomous device, the crew would be eliminated and it would then be possible to operate a Flying Junior solo. Using an inertial measurement unit, the heel angle of the boat in the water can be measured. By setting a desired heel angle on a control panel, a mass driven by a motor can be moved laterally along a track in order to adjust the heel angle accordingly. The goal of this project is to allow a sailor to safely operate a Flying Junior dinghy without a crew and be able to maintain full control of the boat. The project objectives include:

- Design a weighted mechanical system to replicate the righting force of a 150 lb. crew
- Adjust a moving mass along a track to correct the heel to the desired angle
- Create a device that weighs less than 150 lbs.
- Provide enough energy for multiple hours of continuous use.

The end result of this project is a device that can be easily attached to any Flying Junior dingy, weighs less than an average human being, and can efficiently maintain a specific heel angle as well as an experienced crew is able to.

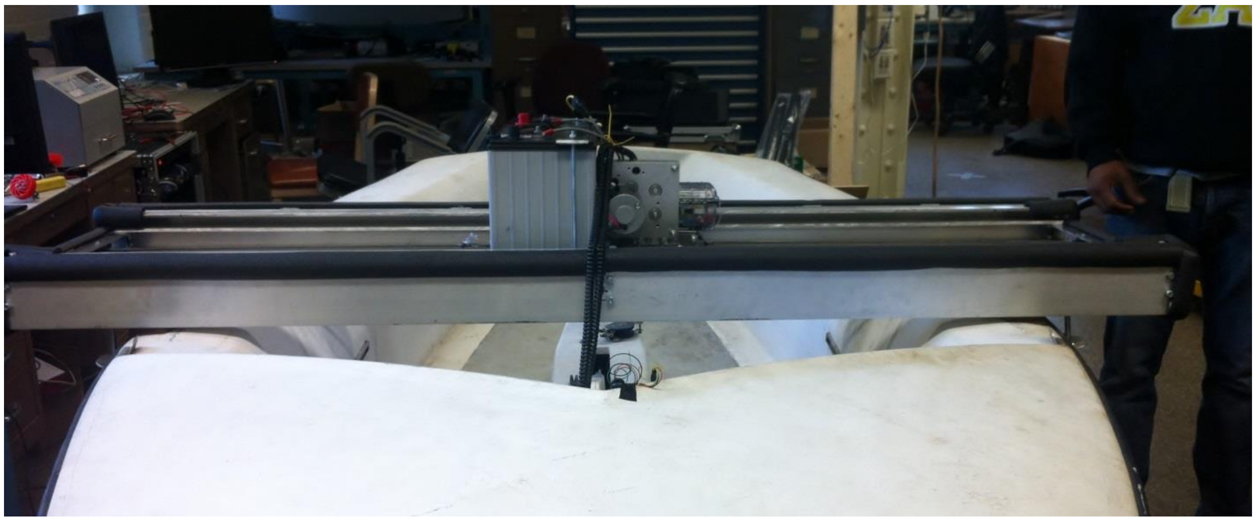


Figure 1: Sailboat Stabilization System

2.0 Literature Review

This chapter functions to deliver the background necessary in order to research and develop an autonomous stabilization system for a sailboat. First we discuss sailing, including sailboat boat terminology, the basics of sailing, and the physics of sailing. Second, we focus on the stability of a sailboat, introducing gravitational and buoyant forces as well as the stability curve. Lastly, we focus on stabilization systems, both passive and active, employed on boats today.

2.1 Sailing

To understand the scope of this project, one must first know some key aspects of sailing, including the parts of a sailboat, how to sail a vessel, and how one is able to sail. The world of sailing has its own unique set of terms to describe the various parts of sailboats. These are the terms that will be used throughout the report so it is important to become familiar with them.

2.1.1 Sailboat Terminology

Sailboats are complex pieces of engineering that have been finely tuned over their several thousand years in existence in order to optimize utilization of wind speed and direction. There are many styles of sailboats with different quantities, shapes, sizes, and configurations of sails that are specialized for different wind and ocean conditions. The type of sailboat involved in this project is known as a sloop rig. This type of boat has a single mast with two sails. The mainsail is located aft of (behind) the mast and is also connected to a boom, a free swinging horizontal beam that controls the angle of the mainsail. The foresail is called the jib and is located at the front of the boat. The purpose of the jib is primarily to increase speed and improve handling of the boat. A sloop rig requires two people to operate. In the stern of the boat is the skipper. This person controls the angle of the boom and the rudder and has almost complete control of the operation of the boat. The other operator, located toward the bow, is known as the crew. The primary need for this person is to use their weight to influence the roll of the boat through a method called hiking. Hiking is essentially the act of leaning out of the boat so as to shift one's center of gravity away from the center of rotation of the boat in order to generate a torque that opposes the torque created by the wind on the sails. The crew adjusts his/her hiking position in order to achieve the desired angle of heel. In addition to this, the crew is also responsible for adjusting the tension of the jib.

The bottom of the sail is known as the foot and on the main sail it is connected to the outhaul, which is the line used to control the tension of the lower section of the sail. The top of the sail is the head; this is connected to the top of the mast with the halyard, which is the line used to hoist the sail to the top of the mast. The leading edge of the sail is the luff which is tensioned by a line called the Cunningham. The tailing edge of the sail is the leech, and in some cases is tightened by a leech line to prevent fluttering of the sail. In addition, the boom-vang is a line that effects the leech because it controls the vertical angle of the boom which pivots at a set point on the mast. This tightens the whole sail, but because it is pivoting off of the mast the tail edge experiences the most displacement, thus the most tension. The leading bottom corner of the sail is the tack; this is connected to a fixed point for both the mainsail and the jib. The aft bottom corner of the sail is the clew; in the case of the jib, it is the point of the sail that is not connected to any fixed point and is used to control the boat. The foot of the mainsail is attached to the boom which is attached to the mainsheet, the line that controls the trim of the sailboat. A line is the term used to describe a rope that is in use. The word ‘rope’ is only used for a rope that is not currently in use or serving any purpose. In addition, the term sheet is used to describe a line that is used to control the movable corners, or clews, of the sails. The main body of the boat is known as the hull; the front of the hull is the bow, while the rear is the stern. The left and right sides of the boat are referred to as port and starboard respectively. At the stern, there is a rudder that is used to steer the boat. A tiller is attached to the rudder to control its angle relative to the boat, thus controlling its heading. Protruding from the bottom of the center of the boat is the centerboard which is a foil fixed in place that helps greatly in stabilizing the boat. Figure 2 provides visualization for parts of the sailboat mentioned above.

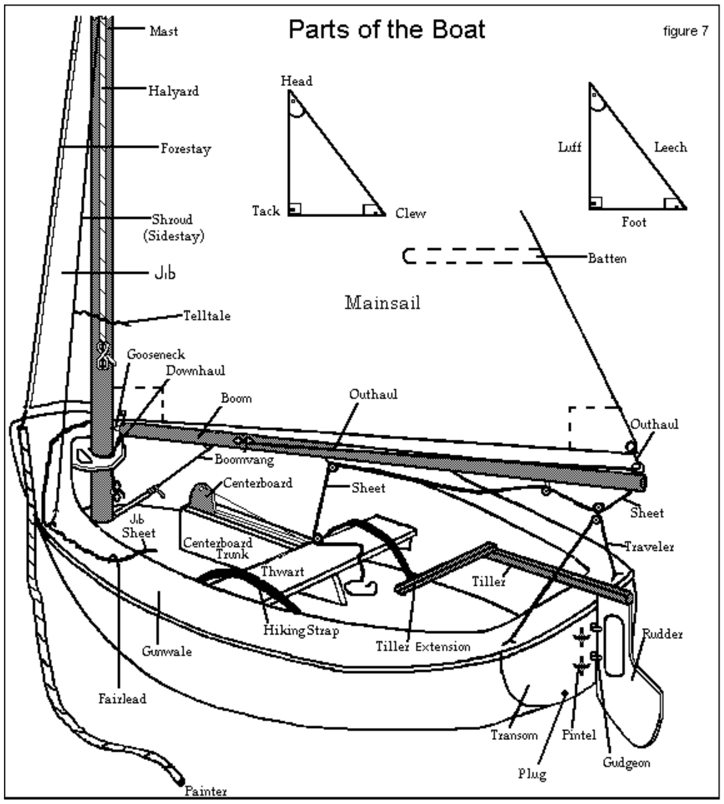


Figure 2: Parts of the Sailboat (Kylander, 2013)

2.1.2 Basics of Sailing

An important aspect in sailing a boat is the orientation of the sailboat relative to the wind. Figure 3 below shows the seven general positions a sailboat can travel relative to the direction of the wind.

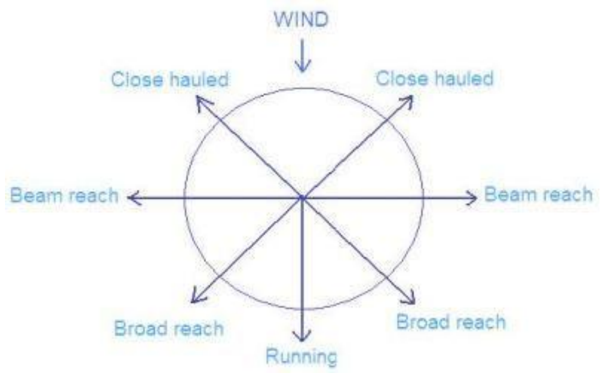


Figure 3: Orientation of Boat Relative to True Wind (Lochhaas, 2013)

As shown in Figure 3, one cannot sail directly into the wind; however, the sailboat can travel at approximately 45° against the wind, commonly referred to as close hauled. Sailing perpendicular to the wind's direction is known as beam reach, with the wind coming directly over either side of the beam, the widest part of the boat. Broad reach is acquired when the sailboat is oriented 135° relative the wind's direction. When the wind is blowing directly in line with the sailboat, also known as running, the sailboat travels in the same direction as the wind.

The motion of a sailboat affects the relative wind direction and magnitude because the boat's movement through the air creates its own relative wind. This idea can be seen onshore as well by simply running. As a runner moves directly into the wind, the wind seems much stronger than when he was standing still. This is because the apparent wind is the vector sum of the inverse of velocity forward and the actual wind. Figure 4 below shows the wind speed relative to the boat in red, the boats velocity in blue and the true wind in green. One can observe that when a boat is running from the wind the apparent wind is low, because the true wind vector and velocity vector oppose each other.

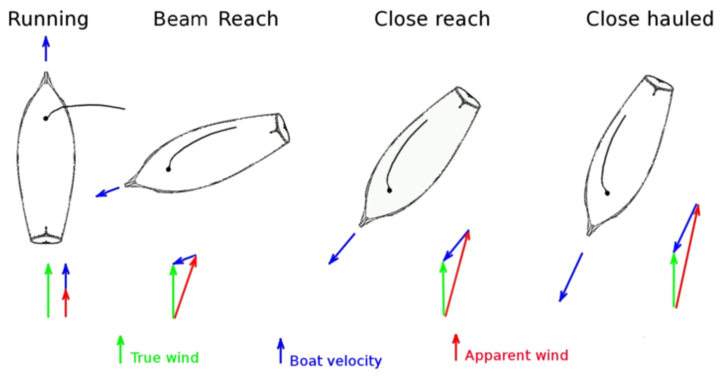


Figure 4: Boat Velocity relative to True and Apparent Wind (Erwan, 1972)

For a boat to travel in a forward motion, the sails must be adjusted to provide lift and drag forces. The contact of the wind and the sail creates a roll force that makes the sailboat rotate around the centerline of the boat, also known as heeling. The sailor can counteract this force by positioning their center of mass on the windward side, stabilizing the roll of the sailboat. To increase the speed of the sailboat, one must pull the sheets of the mainsail and jib, bringing the corners closer to the boat and allowing the wind to fill the sail. Pulling the sheets until the sails stop luffing, oscillations in the shape of the sail due to the wind traveling on each side, allows the force produced by the sails to move the boat forward. The motion of adjusting these sheets is known as trimming, allowing the sail to take the best shape for the direction you are sailing relative to the wind. When trimming, the sail must be tight enough so that the luff of the sail is not flapping; however, an over tightened sail only allows the wind to blow against one side, causing the boat to heel over. Trimming is commonly associated with the turning of a boat relative to the wind; as a boat turns towards the wind (heading up), the sails are pulled in and when a boat turns away from the wind (heading down), the sheets are eased out.

In order to turn the boat towards port, the tiller must be moved to starboard. Moving the tiller in one direction rotates the rudder to the other side, where the resulting forces create a turning moment about the center of the boat. Moving from starboard tack to port tack, as depicted in Figure 5, involves two motions: turning the boat and either tacking or gybing. These two terms describe the manner in which the sail crosses over to the opposite side of the boat.

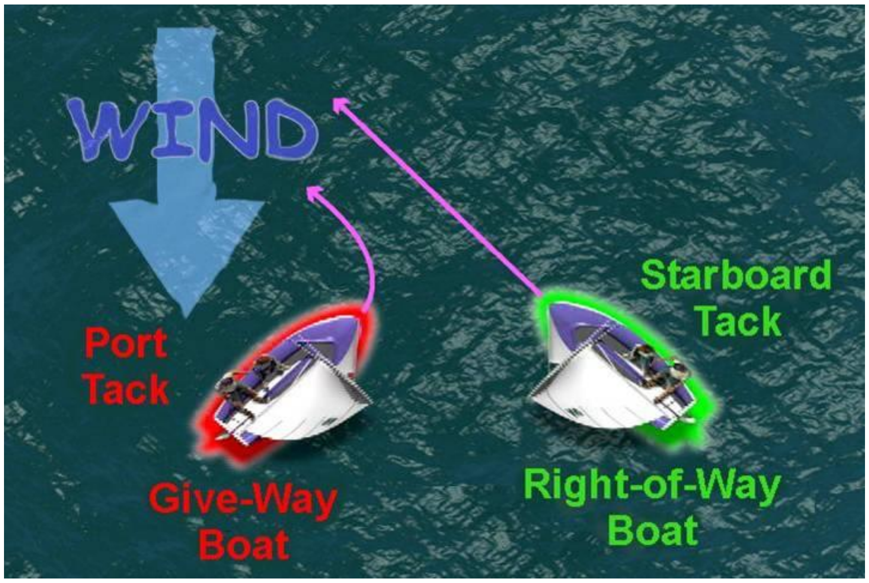


Figure 5: Port Tack and Starboard Tack (Northern Light, 2009)

These are two distinct turns: turning 90 degrees through the wind from close hauled to close hauled (tacking) and turning so that the wind crosses the stern, a run to a run (gybing). Tacking involves oncoming wind while gybing occurs going down wind and differs from tacking in that the sails and boom move from one side to the other very aggressively. This is because before gybing the boat is on a run, where the sails are fully out. When the wind begins crossing the opposite side of the boat, the boom must travel a large distance across the boat and can reach dangerous speeds. Gaining speed before tacking ensures that the boat will not stall when turning into and through the wind. As the boat crosses the wind and stops heeling, the crew must shift their weight to the opposite side. Additionally after a tack or a gybe, the jib sheet must be moved to the other side, similar to the mainsail.

2.1.3 Physics of Sailing

Sailboats are able to move forward as a result of the interactions between the sails and the wind as well as the interactions between the centerboard and the water. The four forces which primarily affect the boats movement are the following: the force of the wind on the sail, the force of water on the centerboard and rudder, the buoyancy force on the hull, and the hiking force from the boat’s skipper and crew.

There are two main components of the force exerted by the wind, the drag component and the lift component. These components maintain a constant direction relative to the wind but the magnitude of each varies. The lift component comes from the difference between the air pressures on each side of the sail. This difference is attributed to Bernoulli’s principle, which states that for an ideal flow, an increase in the speed of a fluid occurs simultaneously with a decrease in pressure or a decrease in the fluid’s potential energy. The air travels at different speeds over each side of the sail due to its shape, as seen in Figure 6 below.

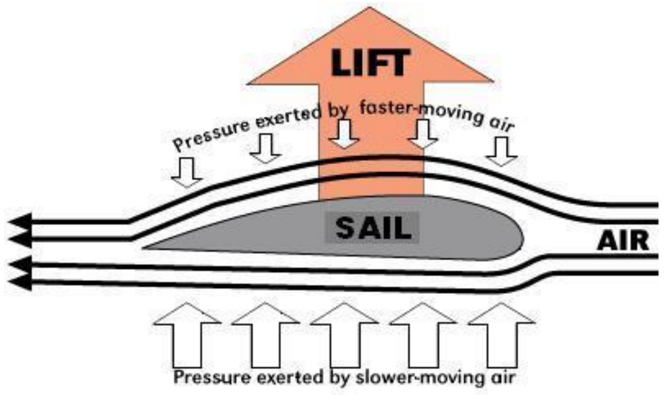

LIFT
Pressure exerted by faster-moving air
SAIL
AIR
Pressure exerted by slower-moving air

Figure 6: Pressure of Air on Sails (Thriveni, 2007)

The air must travel on both sides of the sail in an equal amount of time; however, the distance traveled on each side is different, resulting in different air speeds. Therefore, the air traveling on the convex side of the sail travels a larger distance at a faster speed, but in the same amount of time. This side of the sail now has reduced pressure so a pressure driven lift force is created. This resulting lift force occurs perpendicular to the wind and varies in magnitude. The drag is the force exerted on the sail to push it downwind. This force is parallel to wind and is dependent on the aerodynamic properties of the boat, rig and sails.

The lift force combined with the drag force produce the resultant force, as shown below. This resultant force is the overall force due to the wind and when trimmed correctly is perpendicular to the sail at its centroid. By knowing this force, it is possible determine the force driving the boat forward as well as the force causing the boat to heel**.** Figure 7, shows how these forces work when traveling upwind on a close hauled course.

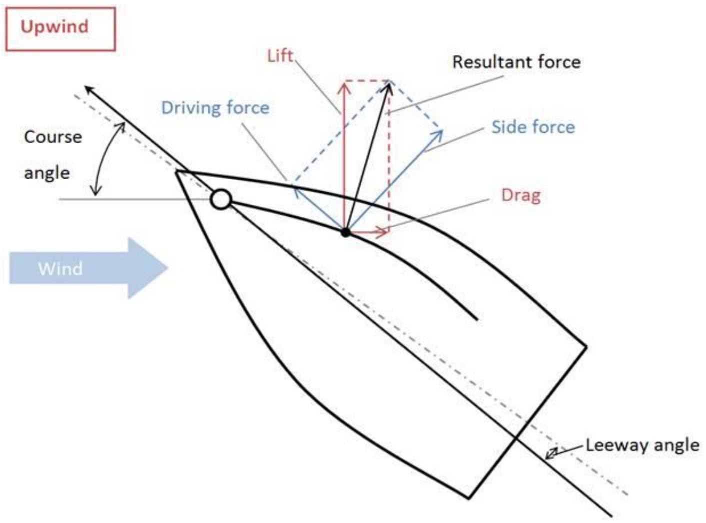


Figure 7: Resultant Force Vectors (Lafforgue, 2007)

Figure 7 illustrates how the resultant force, perpendicular to the sail, is the vector sum of the drag and lift forces. Figure 8 below shows how adjusting the sail relative to the wind affects the resultant force. For example, going directly with the wind is driven predominantly by the drag force, whereas close hauled; the main component is the lift force.

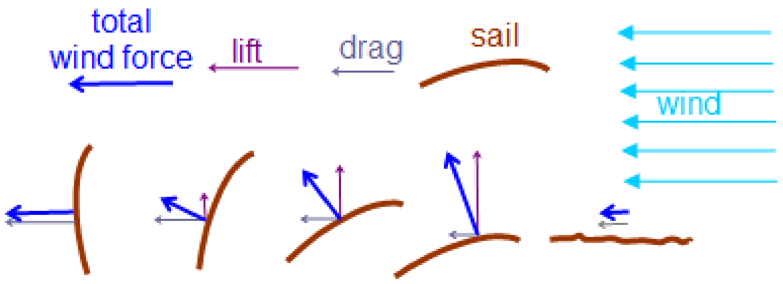


Figure 8: Sail Adjustments Relative to Wind (Salu, 2009)

As the boat travels through the water, it experiences a drag force in the direction of the oncoming water and a perpendicular lift force from the water traveling over the centerboard. When a boat is traveling at a constant speed, the resultant force of the water acting on the centerboard is equal and opposite of the force on the sails as depicted in Figure 9 below. The combination of the forces from the wind and forces from the water creates forward motion.

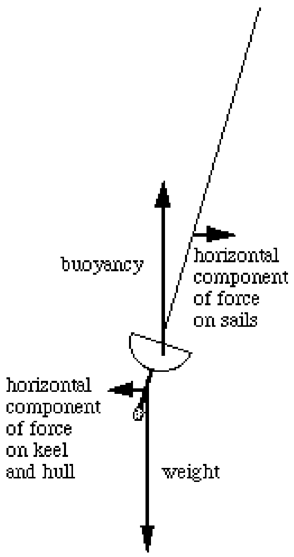

buoyancy
horizontal component of force on sails
horizontal component of force on keel and hull
weight

Figure 9: Forces Acting on Sailboat (Wolfe, 2002)

The boat rotates primarily around the centerline of the boat from bow to stern. This rotation is caused by the forces discussed above as well as the buoyancy of the boat and the weight of the crew. As the boat heels over, the center of buoyancy moves further from the centerline of the boat and provides a stabilization force. However, as the boat heels, the effectiveness of the crew begins to decrease, as shown in Figure 10.

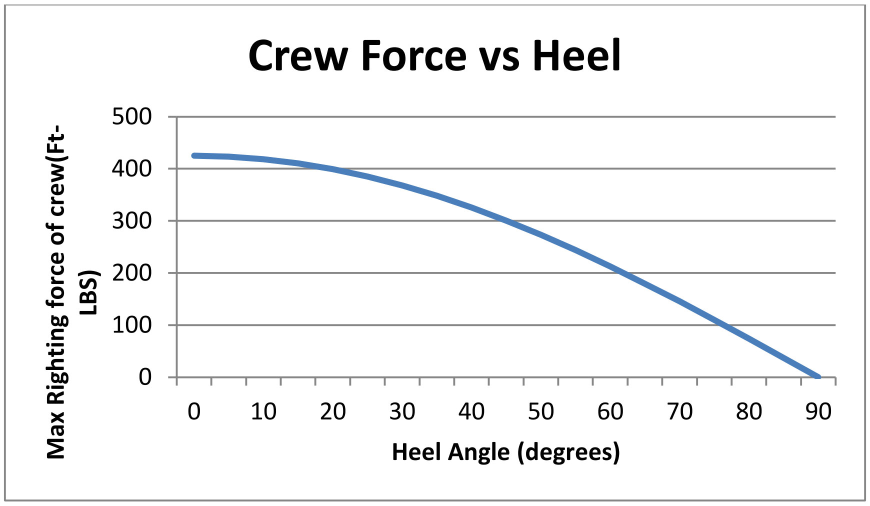


Figure 10: Crew Force vs Angle of Heel

The crew uses their weight to maintain a desired heel angle, one of their main roles. As the boat heels, the force component perpendicular to the lever arm decreases due to downward gravitational force.

2.2 Stability

Archimedes principle, the physical law of buoyancy, states that any body completely or partially submerged in a fluid (gas or liquid), at rest, is acted upon by an upward, or buoyant, force, the magnitude of which is equal to the weight of the fluid displaced by the body. These gravitational and buoyant forces are those which act upon the boat at rest, neglecting the external forces, such as wave motion and wind, which significantly decrease stability at sea.

2.2.1 Gravitational and Buoyant Forces

There are three types of equilibrium that can occur on a sailing vessel: stable, neutral, and unstable. Stable equilibrium implies that the boat will return to its original position, having positive stability, if acted upon by an external force. Neutral equilibrium implies the boat will remain in its displaced location after acted upon by an external force, having the characteristic of neutral stability. Unstable equilibrium occurs when the boat continues to move in the same direction of the external force and continues to do so after the force has been removed. For this case, the boat is initially unstable. (Edwad V. Lewis, 1988)

Gravitational forces are downward forces distributed along the body's length, for which the resultant force is found by using the mass of the body and the acceleration due to gravity (g). The buoyant forces act vertically upwards on the body and can be found by first knowing this gravitational force, which is equal to the weight or mass of displaced fluid for which it is submerged in. Knowing the weight of the body (W), one can calculate the volume of displacement (Δ) using equation 1, where p is the mass density of the fluid and g is the acceleration due to gravity.

$$\Delta = \frac{W}{pg}$$

Equation 1: Volume of Displacement

The centroid of the underwater portion of a submerged object is the center of buoyancy, a point at which the resultant buoyant force vertically passes through. The center of buoyancy, the center of gravity for the displaced fluid, can constantly change at sea; however, the center of gravity for the boat is fixed. The center of gravity is the point for which a resultant weight force passes through, which tend to be located near the middle of the body. For the body to float, the buoyant force must be equal to or greater than the gravitational force. For this reason, a designer must take into account any content which will be added onto the vessel to achieve the proper distance of the bow to the waterline to minimize resistance.

The interaction of gravitational and buoyant forces determines the attitude of the object. If no other forces act on the object, the object will settle once the buoyant force is equal to its weight, and will continue to rotate until two conditions are met. First, the center of buoyancy and center of gravity must lie on the same vertical axis. Second, any rotation from this position will cause equal gravitational and buoyant forces to rotate the object back to its original position, resulting in stable equilibrium. (Edwad V. Lewis, 1988)

2.2.2 Stability Curve

In order to keep a boat stabilized, it is important to understand the stability curve. The stability curve shows the relationship between the center of gravity and center of buoyancy of a boat. The center of gravity of the boat always remains the same; however the center of buoyancy is dependent on the volume and relative location of air displaced below sea level by the hull of the boat. The location and magnitude of this moment varies based on the heel of the boat since the hull is not perfectly circular. The righting moment, which is the force the boat is exerting toward a vertical orientation, is greatest when the center of gravity and center of buoyancy are at a maximum horizontal distance. In the example in Figure 11, this occurs at 60 degrees from vertical.

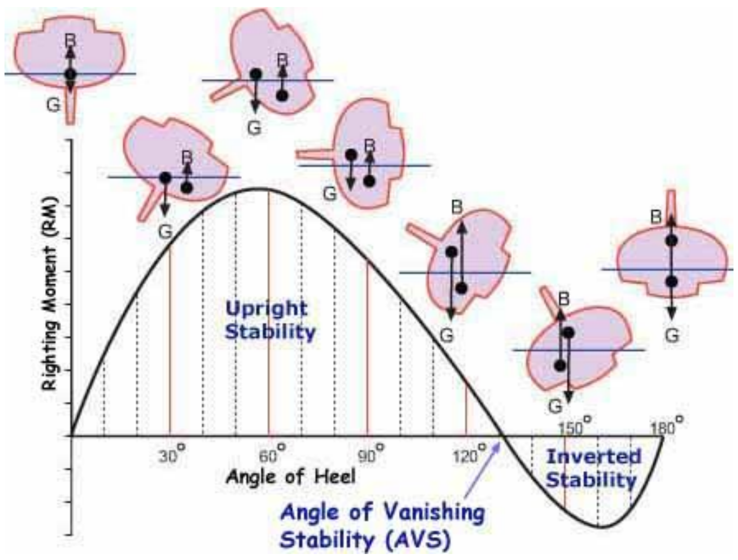


Figure 11: Stability Curve (Simpson, 2014)

This angle is known as the Angle of Maximum Stability (AMS). As the angle increases past this point, the righting moment decreases. The Angle of Vanishing Stability (AVS) is the point at which the relationship between center of gravity and center of buoyancy tend to capsize the boat rather than right it. The boat then enters a state of inverted stability that follows the same dynamics as upright stability.

2.3 Stabilization Systems

Both active and passive systems are employed in stabilizing marine vessels due to the effects of gravitational, buoyant, and external forces. Active systems require power to operate the system while passive systems do not, consisting of only mechanical components.

2.3.1 Passive Systems

A bilge keel, as shown in Figure 12, is a passive system added onto the boat to reduce the amount of roll on the boat. Primarily, two bilge keels are welded to each side of the craft, toward the lower portion of the hull to decrease the draft of the vessel. The purpose of the bilge keel is to increase the hydrodynamic resistance to roll; however, bilge keels tend to increase hydrodynamic resistance to forward motion (Kasten, 2012). Although this is a fairly simple and relatively inexpensive addition to a boat, it slows down forward motion and its effectiveness is far less than most other methods of stabilization. Bilge keels are often used on smaller fishing boats.

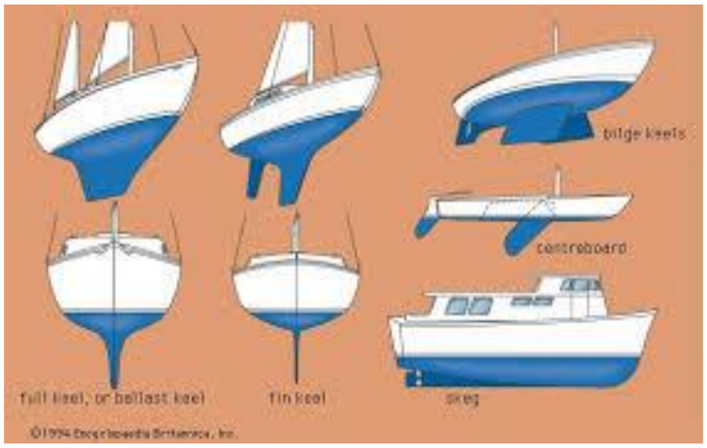

bilge keels
centreboard
full keel, or ballast keel
fin keel
skeg
©1994 Encyclopaedia Britannica, Inc.

Figure 12: Examples of keels (Encyclopedia Britannica, 1994)

Another passive system involves an outrigger, which is incorporated into the boat's rigging on the main hull. The outrigger extends beyond the side of the boat, acting as another hull parallel to the main hull used to reduce roll and increase stability (Kasten, 2012). Depending on the size of the outrigger and the utilization of either one or two outriggers, they are extremely effective in maintaining stability of a boat. The downsides are that they are quite expensive, large, and heavy, creating large amounts of drag and rendering the efficiency minimal. For these reasons, they are rarely used on monohull sailboats.

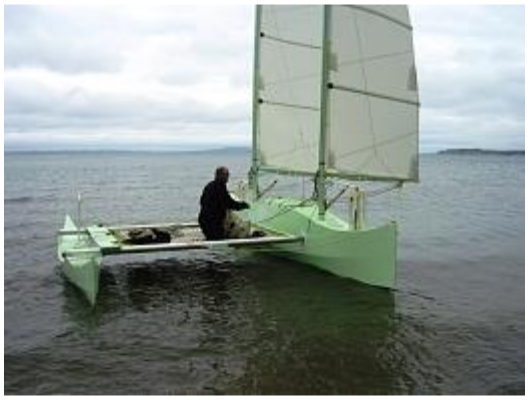


Figure 13: Outrigger on Sailboat (Tenacity, 2005)

Anti-roll tank systems incorporate tanks fitted into the ship to stabilize roll motion. Commonly known as Flume tanks, free surface tanks run the entire width of the vessel, where liquid will move from side to side in response to roll. Each tank is fitted with several baffles to slow the rate of water transfer from the port side to the starboard side. U-tube tanks also span the entire width of the vessel; however, a crossover duct and an air column connect the tanks. The partially filled tanks create a gap of air on top of each tank for fluid to flow from tank to tank as the boat begins to roll. This system utilizes less space than free surface tanks because the space above and below the cross-over duct is available. Lastly, external stabilizing tanks, used in the early 1900s, consisted of the same concept used in U-tanks; however, the two tanks are only connected by the air column. Water flows into and out of the tanks through an opening in the ship's hull to the sea. This system is susceptible to high levels of corrosion due to sea water and hydrodynamic resistance. The resistance to forward motion is created by the holes in the hull and momentum drag force, the force required to drive seawater outside the boat to the speed of the boat as the water moves in. Antiroll tanks can be large and add significant weight and drag to small boats. They are intended to only reduce roll motion so if the boat is at a constant heel, the antiroll tank becomes less effective as water slowly shifts to the lower side of the tank, eventually rendering it useless and even detrimental to the boats stability. (Edward V. Lewis, 1988)

Paravanes, as shown in Figure 14, are mainly used to stabilize trawlers and slow moving vessels. They are essentially weighted water kites dragged underwater on both sides of the boat, hanging from an outrigger pole to well below the surface of water. As these weights are pushed through the water, they resist being pulled up through the water by the rolling movements of the boat (Kasten, 2012). Paravanes are very effective at stabilizing a boat, and are relatively cheap and easy to install. Despite this, they are rarely used on sailboats as their physical configuration often interferes with the operation and performance of the sails.

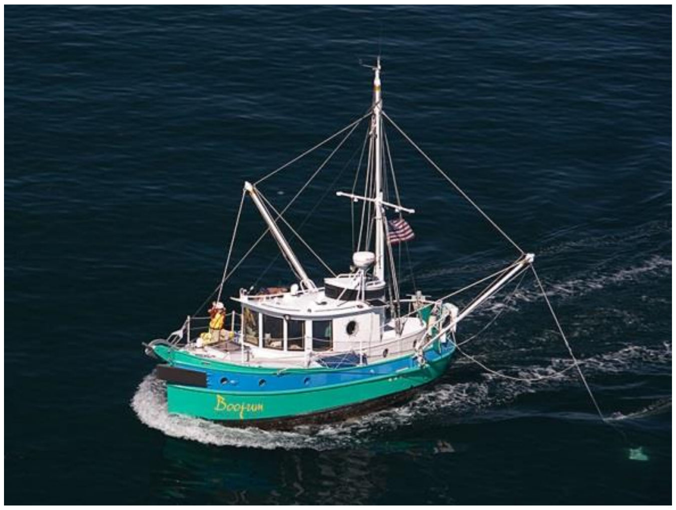


Figure 14: Paravanes on a Boat (Kasten, 2012)

2.3.2 Active Systems

As compared to the passive stabilization systems, an alternative method widely used today involves the application of active stabilization. Active stability systems are defined by the need to input energy to the system in the form of a pump, hydraulic piston, or electric actuator (Wikipedia, 2013). Many vessels today are fitted with active stability systems which can range from stabilizer fins, U-tube tanks, and even gyroscopic inertial stabilizers. With the harsh conditions and effects on ships out at sea, the use of these stabilization methods proves to be a necessity and a solution for many ongoing mariners.

Stabilizer fins can be helpful in reducing the effects of roll on a ship. They are typically mounted beneath the waterline and work by producing lift or down force when the vessel is in motion. Furthermore, these devices can be used with gyroscopes to change the angle of the fin in order to counteract roll. When the ship begins to roll to the left or right, the fins counter that motion and keep it upright. This technology has made marine travel easier, calming the fear of a vessel turning over. Active stabilizer fins are extremely efficient at attenuating roll. The efficiency is a function of velocity, so they become more effective as velocity of the vessel increases. Unfortunately stabilizer fins are quite expensive, difficult to install and rarely used on small vessels. (Sheng, Liang, Gao-yun, & Bing, 2008)

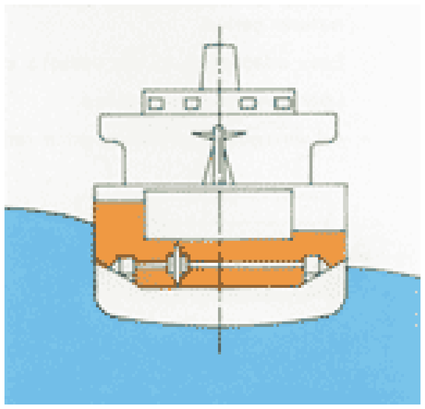


Figure 15: Active Stabilization on a Boat (Foure Lagadec, 2006)

Active tank stabilizers use an axial flow pump and servo-controlled valve system to force the water from one side of the ship to the other rather than allowing it to move freely. Furthermore, they are used to control the flow of air and liquid throughout the tanks. The main disadvantage to this system is when the pump is operated there is a time lag for a sizeable amount of fluid to arrive at a tank, limiting instant stabilization, imposing and issue for small vessels.

Flywheel gyroscopes are another technology used in active stabilization today. This device is used to measure or maintain balance based on the principle of angular momentum. A gyroscope is comprised of three different axes: spin, input, and output. The spin axis is the axis about which the flywheel is spinning and is vertical for a boat gyro. The input axis is the axis about which input torques are applied (Townsend, Murphy, & Shenoi, 2007). For a boat, the principal input axis is the longitudinal axis of the boat since that is the axis around which the boat rolls. What this means is that the gyro will rotate about this longitudinal axis also known as a transverse axis in reaction to an input. This system functions similar to a spinning top. As the rotational speed increases, the tendency to stand straight and maintain an upright position increases. Flywheel gyroscopic stabilizers are a very effective stabilization method, but they are very heavy, expensive, have high power consumption, and take a significant amount of time to spool up to operating RPM (Wikipedia, 2013).

It is clear that for small sailboats, there are very few viable options for effective stabilization. There is a strong need for a stabilization system that is easy to install, light-weight, and inexpensive.

3.0 Methodology

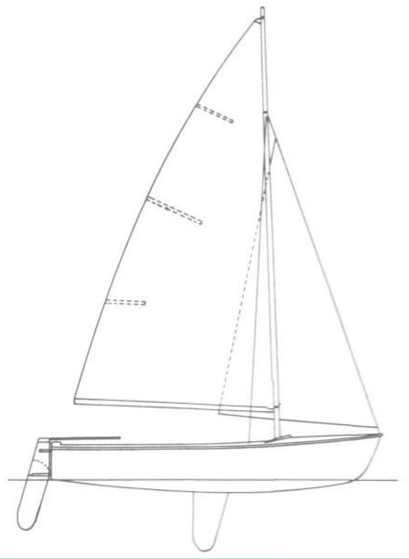
The objective of this project is to develop an autonomous stabilizations system for a Flying Junior sailboat. First, we performed a mechanical analysis to determine the physical characteristics of the sailboat and to calculate the forces and torques necessary to replace the crew. Next, we analyzed the power requirements and determined the motor required to perform the tasks set forth from the mechanical analysis. We also investigated the types of sensors that are required to provide the necessary information to control the system. Finally we calculated what size battery is necessary to provide a long duration of use as well as provide sufficient power for the circuitry.

3.1 Mechanical Analysis

This section covers the process of examining the mechanical aspects of the system. This entails the analysis of the boat's physical characteristics, examination of the role of a crew, and desired mechanical properties of the system.

3.1.1 Boat Analysis

The boat we have selected to design this system around is the Flying Junior Dinghy. This boat has low stability and relies on two members to keep the boat flat by hiking. This project aims to eliminate the second crew and to allow for solo sailing. In order to accomplish this, it is vital to examine the forces acting on the system to determine how the system responds without any crew.



Hull Type:	Centerboard Dinghy	**Rig Type:**	Fractional Sloop
LOA:	13.25' / 4.04m	**LWL:**	12.25' / 3.73m
Beam:	5.25' / 1.60m	**Listed SA:**	100 ft^2 / 9.29 m^2
Draft (max.)	2.50' / 0.76m	**Draft (min.)**	0.58' / 0.18m
Disp.	209 lbs./ 95 kgs.	**Ballast:**	

Figure 16: Flying Junior Characteristics (Browning, 2014)

3.1.2 Force and Torque Calculations

The main driving force of the system is the force caused by the wind, which can be calculated for a specific sail area and wind speed. Using an apparent wind speed of 15 knots, which is at the higher end of comfortable sailing conditions, and a sail area of 100 square feet, which is the combined area of a typical FJ mainsail and jib, it is possible to determine the forces acting on the boat. The magnitude of the force can be calculated using the equation:

$$\text{Sail Force}=0.0034 \times \text{Sail area (ft}^2\text{)} \times \text{Force Coefficient} \times \text{Wind Speed}^2 \text{ (knots}^2\text{)}$$

The equation above is given in Royce's Sailing Illustrated sailing bible (Royce, 1993). The average force coefficient for similar boats is 1.1 and varies depending on the sail and rig design. It can also be noted that the wind speed is squared, resulting in an exponential increase in sail force with a linear increase in wind speed.

Sail Force = 0.0034 x 100 x 1.1 x 15^2 = 84.2 lbs.

This is the force applied at the centroid of the sails, which was calculated using SolidWorks software. This was accomplished by modeling the two sails in an appropriate relation to each other and determining the center point as shown in Figure 17.

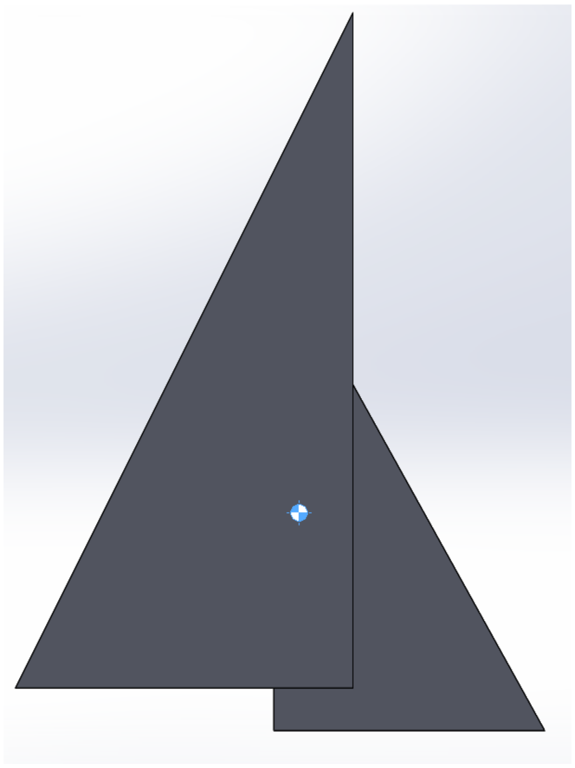


Figure 17: Centroid of Sails

Before we are able to determine the torque about the centerline caused by the wind we must determine the position of the boom relative to the centerline. The maximum torque occurs on closed hauled so the boom position was calculated for this boat on the ideal close hauled position, in which the tip of the boom is over the air tank of the dinghy. The boom angle is calculated by using the boom’s length and the distance of the tip of the boom from centerline, resulting in a boom angle of 18.2 degrees. From this it is possible to determine the force on the sail perpendicular to the lever arm. The resulting force is 80 lbs. applied to the centroid as shown in Figure 14. This force is equal and opposite to the force generated on the centerboard of the boat. The centroid of the centerboard was also used for the location of the force from the water. The torques were examined about the centerline of the boat at the level of the rail as depicted in Figure 18.

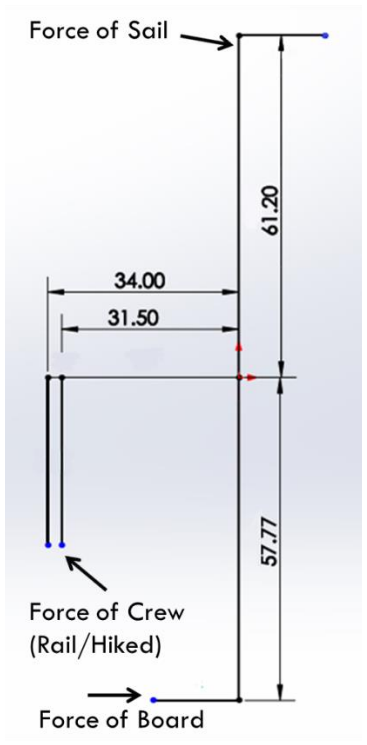


Figure 18: Torque Calculations (*units are in inches)

The following table contains values that were calculated or measured based of the dimensions of the boat.

Source of force	Magnitude of force (lbs.)	Lever arm (Ft)	Torque about centerline(ft-lbs)
Force of sail	80	5.1	408
Force of board	80	4.81	385
Force of skipper (on rail/hiked)	150 150	2.63 2.83	-395 -425
Total Torque (on rail/hiked)			398 368

Table 1: Table of Forces

This table is based off of the boat under the following conditions; the absence of the crew, a level boat, close hauled, sails trimmed correctly, 15 knots of breeze, and a 150 lbs. skipper who is 5’ 8” tall. It is clear that at this angle it is possible to hold the boat steady with 398 ft-lbs of force. If the system can provide the same torque as a crew hiked out (425 ft-lbs) it will be capable of maintaining this heel angle.

3.2 Power Analysis

Delivering power to the stabilization system is a challenging aspect of this project. We will need to calculate the power consumption of the computer and motor that we will be selecting and determine how large of a battery will be required to adequately power the system for a specified amount of time. We will need to take into consideration the weight of the battery, as weight is a crucial limiting factor for this system. The goal is to keep the weight minimal, while still having enough battery capacity to allow for several hours of continuous operation. Depending on the final weight of the stabilizer, the size of the battery can be adjusted to meet weight specifications. Deep cycle lead acid batteries will be used to power this system since they are extremely durable, have a large capacity, can be regularly deeply discharged, and are relatively inexpensive. Lithium batteries would be a much more lightweight solution; however they are not cost effective for this project.

3.3 Motor Analysis

From the power analysis, we calculated that the motor needs to output at least 145 watts in order to handle moving a 70 lb. load up a 30 degree frictionless incline. The motor suggested to accomplish this is the CIM motor. This motor has a maximum power output of

337 Watts and runs off of a 12V power supply. This motor has a 2.5 inch diameter and a 4.34 inch body and weighs only 2.8 lbs. This motor fits all of our power requirements, will be able to handle the required load extremely well, and is also very cost effective.

The motor should not be active when the system is at rest so a braking system will need to be implemented. A brake must be able to hold the mass at any position along the track without it shifting and should be able to actuate and release quickly in order to allow for movement to resume. One way to achieve this is with a friction brake that clamps to the track and holds the mass in place. Another solution would be a disk brake attached to the motor shaft. A third method of braking would be a pin that actuates and is inserted into a slot in either the track or a disk attached to the motor. Any of these solutions would effectively stop and hold the mass in place; however, the first two braking methods using friction would require significantly more power than the actuated pin. Conversely, the actuated pin method would require significantly more precision in order to line up the pin and slot reliably.

3.4 Sensor Analysis

This section covers the sensors needed to provide information on the orientation of the boat, position of the mass, the position of the boom, end of track sensor, and skipper controls to adjust and control the system.

3.4.1 Orientation of the Boat

This system requires the use of an Inertial Measurement Unit (IMU) in order to properly measure and calculate the orientation of the boat. One variation of this sensor to be analyzed are Micro-Electro-Mechanical Systems, or MEMS (Esfandyari, De Nuccio, & Xu, 2012). This is a technology that in its most general form can be defined as miniaturized mechanical and electro-mechanical elements that are made using the techniques of micro fabrication in this projects case, the 9 Degrees of Freedom (9DOF) Razor IMU from SparkFun Electronics. The 9DOF Razor IMU incorporates three sensors, an ITG-3200 (MEMS triple-axis gyro), ADXL345 (triple-axis accelerometer), and HMC5883L (triple-axis magnetometer) to give you nine degrees of inertial measurement. The outputs of all sensors are processed by an on-board ATmega328, a high-performance Atmel 8-bit AVR RISC based microcontroller, and output over a serial interface (Encyclopedia Britannica, 2013). This enables the 9DOF Razor to be used as a very powerful control mechanism for UAVs, autonomous vehicles, and in this projects implementation for stabilization systems. Figure 19 shows the 9DOF Razor IMU which will be used for the purpose of this projects initial testing.

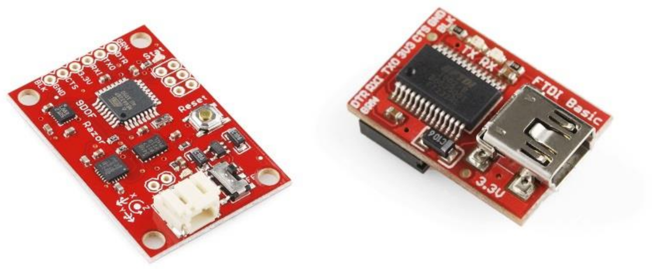


Figure 19: 9DOF Razor IMU (left), FTDI Breakout Board (right) (SparkFun, 2014)

The 9DOF IMU operates at 3.3V DC and any power supplied to the white JST connector will be regulated down to this operating voltage. The output header is designed to mate with the 3.3V FTDI Basic Breakout board, for the purpose of easily connecting the board to a computer's USB port. Additionally, the board can be connected to the Bluetooth Mate or an XBee Explorer for wireless or blutooth applications. In the case of this project, the 5V port is used, which is equipped with a step down converter to properly provide the 3.3 V needed to power the IMU. For more information on the individual characteristics of the sensors, the IMU, or the FTDI see Appendices A, B, and C which includes detail spec sheets on each sensor.

The use of the IMU is beneficial to this project it is being used as a sensor for angular acceleration while the mass on the track will be responsible for the actual stabilization. To further understand this concept one must understand how the sensor uses the Coriolis Effect in order to give an output. This works through deflection of moving objects being viewed from a certain reference point. In a reference frame with clockwise rotation, the deflection is to the left of the motion of the object; in the case of counter-clockwise rotation, the deflection is to the right. When the Coriolis Effect is detected, the continuous movement of the driving mass will cause a capacitance change ΔC which is picked up by the sensing structure and then ΔC is converted to a voltage signal by the internal circuitry. These electrical signals are what can be converted to digital signals using an analog to digital converted ADC in the ATMega328. When programing the microcontroller, theses bits will be used in deciding thresholds for the mechanical devices such as calibration of the mass position to the angle of roll on the boat.

This gyroscope is essential to the scope of this project since the goal of the stabilizing system is to limit the roll to no more than 15 degrees. As discussed previously, attempting to measure the boats orientation using more than one axis can be difficult, thus limiting the axes to measure only the roll of the boat will minimize this complexity. As the mass moves back and forth, the angle measurements must be quickly relayed to the microprocessor controlling the motor which will provide a smooth counteract against rolling thus providing stabilization.

3.4.3 Position of the Mass

The microprocessor needs to be aware of the position of the mass along the track in order to confirm that the mass is in the proper location. One of the ideas for this accomplishment, is through the use of an incremental rotary encoder will be implemented on the motor shaft that will count the number of revolutions in either direction and associate this number with a corresponding position on the track. This type of optical encoder has a two bit output that allows it to communicate the direction of the motor rotation as well as speed. There are two sets of sensors positioned on opposite sides of a disk as shown in Figure 20 below.

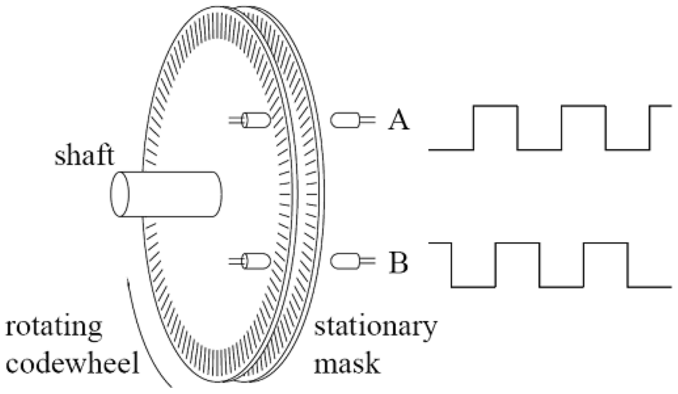


Figure 20: Rotary Encoder (Stephens, 2006)

By offsetting the sensors slightly, the sensor is able to obtain two signals that are out of phase. By determining which signal is ahead of the other, the motor's direction can be determined.

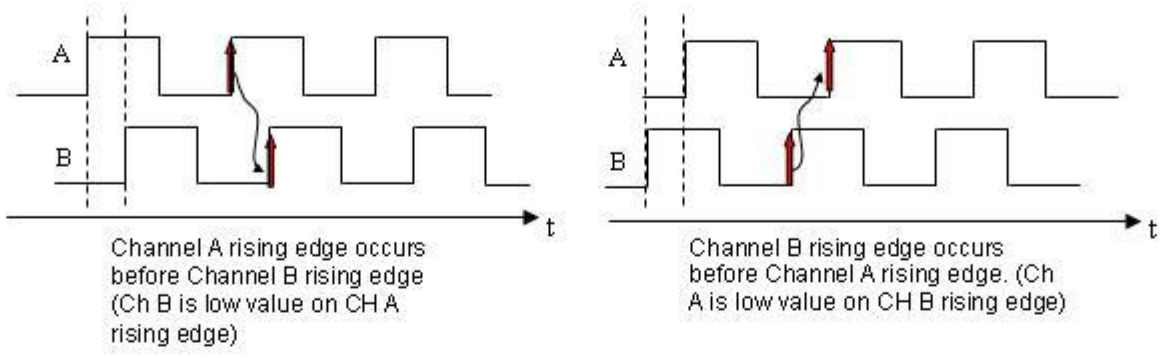

A
B
t
Channel A rising edge occurs before Channel B rising edge (Ch B is low value on CH A rising edge)
A
B
t
Channel B rising edge occurs before Channel A rising edge. (Ch A is low value on CH B rising edge)

Figure 21: Rotary Encoder Two Bit Output (Pepperl+Fuchs, 2014)

Sensing the position of the mass can also be accomplished by using a potentiometer and associating the output voltage to a position along the track.

3.4.4 Position of the Boom

Another sensor that will be implemented is a boom position sensor. A simple switch will be attached to the mast just below the boom that will make contact with the boom as it passes the midpoint to determine what side of the boat the boom is on. This is important to know for tacking purposes. The boom is always leaning toward the lower side of the boat so we can use this information on boom position to prevent the weight from accidentally moving to the lower side of the boat for any reason. The weight will be programmed to only be able to travel along the track on the opposite side from the boom so it will never erroneously put the boat at risk for capsizing. Some amount of hysteresis will need to be programmed into the processing of this input to account for the possibility that the boom remains centered for a significant amount of time and triggers the sensor repeatedly.

3.4.5 End of Track Sensors

It is very important for the system to understand when it has reached the end of the track, so with the use of a toggle switch or button, the microprocessor can stop the movement of the mass in order to prevent it from traveling too far and damaging the system. Bumpers can also be implemented at the end of the track and with the use of the magnetic sensor; the mass on the track can slow down as it approaches the bumper thus deceasing its forward momentum.

3.4.6 Skipper Controls

Located within reach of the skipper will be a panel with several controls that the skipper can use to adjust and control various aspects of the system. Most prominently, there will be a large button that can be depressed to automatically center the mass on the track. This can be used if the skipper does not need any automated assistance, or if a problem arises and the system needs to be disabled. The skipper will also be able to adjust the heel of the boat to a desired angle via a dial on the panel. This is important because different heels are preferable in different conditions. In conjunction with the boom position sensor, the heel will automatically shift to the other side of the boat as the boat tacks. Other controls to adjust various aspects of the system will be added as needed, such as a trim control to adjust for possible drift in the gyroscope.

3.5 Control Analysis

Upon initialization, the system will begin by reading in the position of the mass on the track through the mass position sensor which as described before will be a ten turn potentiometer. Then it should read in the heel set value through the skipper control box which also includes the auto center button. This heel set dial has a limit from 0 to 20 degrees and is labeled for the skipper to easily set his desired heel angle. Next, the system will check to see whether or not the auto center button is pressed. If this condition is met, the mass will be moved to the center position determined by the voltage from the potentiometer. If not pressed, then the system will be in stabilization mode and the microcontroller will read the boom position sensor and select whether or not the boat should be heeled to port or starboard, by negating the heel angle value set by the skipper. Use of these two sensors only requires the skipper to set an absolute heel angle. The microprocessor will determine which tack the boat is on and provide an appropriate righting moment accordingly.

Afterwards, the system will move forward and compare the current heel set value to that of the measured angle from the IMU sensors. This will generate an error value that initializes a point from which the PID controls have to begin error correction. If the new measured angle of the boat is within a threshold of the desired heel angle, then the system loops back into reading the heel set dial and begins the process over again. If the measured angle of heel is not within the threshold of the desired heel angle, then the system will adjust the mass to meet this condition. A visual representation of the overall system design is located in Appendix E of this document.

4.0 Results

The objective of this project was to develop an autonomous stabilizations system for a Flying Junior sailboat. First, we manufactured the mechanical design, including the track system and the active and passive movement systems. Then we constructed our printed circuit board (PCB), potentiometer housing, toggle switch system, push button switch, and watertight housings for all the other electronics. Refer to Appendices F, G, and H to view SolidWorks designs for the track system. Refer to Appendices I and J to view the PCB schematics and board layout.

4.1 Mechanical

The mechanical system of this device consists of a horizontal two-stage bidirectional telescopic slide. The mass is nested inside of a track that allows it to slide from one end to the other on sealed ball bearings. This entire track system is nested inside a secondary track that allows movement of the first track within the second track, also on sealed ball bearings. Movement of the mass within the first stage of the track is controlled by a motor and sprocket that is guided by a chain fixed to either end of the track. Movement of the first stage of the track along the second stage of the track is controlled by a passive line and pulley system that reacts in conjunction to the movement of the mass along the first stage. This then creates simultaneous movement of both stages of the track system. The track itself was constructed from 6061 aluminum alloy and weighs approximately 55 lbs. (without the battery), however it is possible to reduce a significant amount of this weight by cutting material from non-structural areas of the track allowing for greater overall efficiency of the system. With the center of mass extending 5.2 feet out from the center of the boat, the final system is able to create 466 ft-lbs of righting moment, while only weighing 130 pounds.

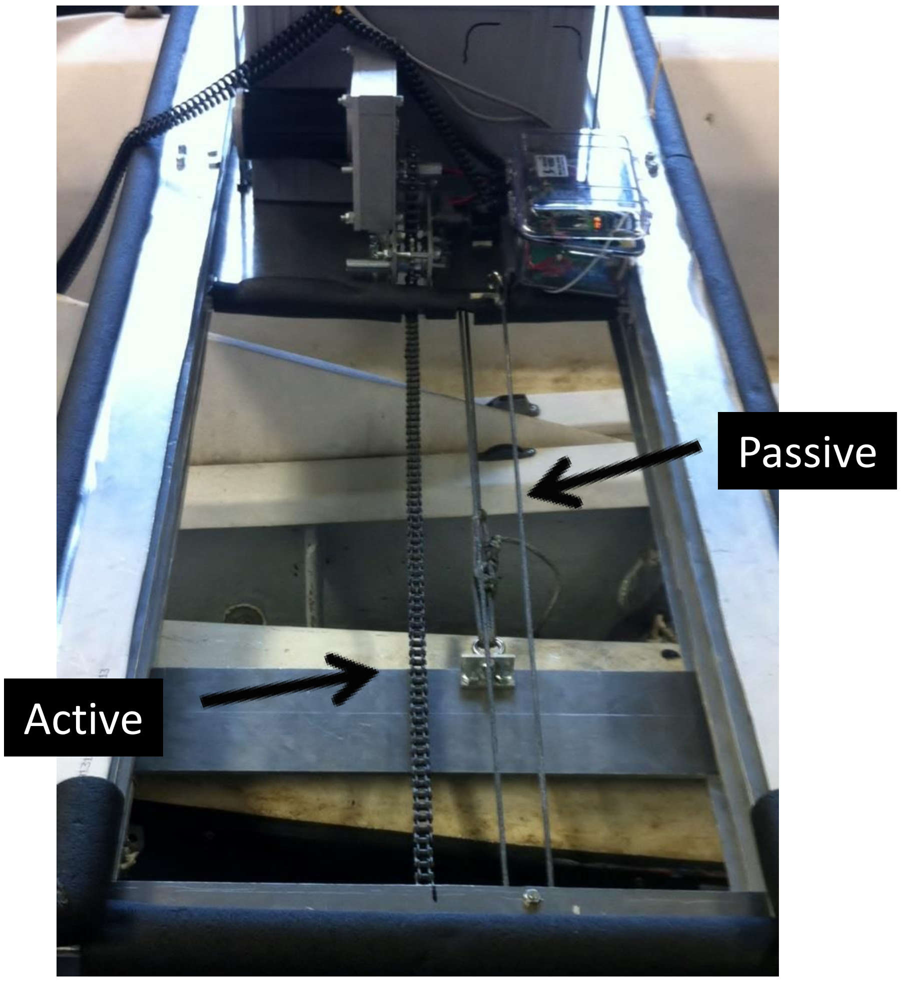


Figure 22: Active and Passive Lines

4.2 Power

The battery used to power the system is a 12 volt, 200 amp-hour deep cycle sealed lead acid marine battery. At an estimated continuous power consumption of 20 amps from the motor and computer, this allows for up to 10 hours of continuous usage. A 40 amp fuse keeps the motor controller and sensitive electronics from experiencing damaging power surges, as the battery is capable of outputting 800 amps peak. For the power to the PCB and signal level inputs, 3.3 and 5 volt regulators were used to step the voltage down to readable signal levels. The battery itself weighs approximately 70 lbs., which accounts for the majority of the 75 lbs. required to achieve the desired torque on the boat.

4.3 Motor

The CIM motor selected to power the movement of the track provided sufficient torque and speed to achieve the desired traversal speed of the mass along the track, even at heel angles steeper than what is normally expected to occur on this boat. At maximum power, the motor is capable of outputting 337 watts. This is greater than the calculated power requirement of 150 watts to move the mass from one end of the track to the other, in roughly 4 seconds, at a heel angle of 30 degrees, 200 watts at 45 degrees, and even 250 watts at 60 degrees. Theoretically the system should still be able to operate at a heel angle of 90 degrees, which requires 290 watts of power, however this has not yet been tested.

Using the TB3 gearbox, as shown in Appendix L with a reduction ratio of 33.8:1 from AndyMark, the speed of the CIM motor was reduced from approximately 4300 RPM to 127 RPM. The sprocket driven by the motor has a circumference of 5.67 inches, requiring 9.2 revolutions for a full traversal of the track, which can be accomplished in just 4.3 seconds.

4.4 Sensors

This section covers the sensors chosen to provide information on the orientation of the boat, position of the mass, the position of the boom, and skipper controls to adjust and control the system. Each component was deliberately chosen to serve its purpose as well as work well in a wet environment and maintain durability for this project.

4.4.1 Orientation of Boat

This project requires an IMU to measure and calculate the orientation of the boat. Although the 9DOF Razor could accomplish this, it does not have any input/output (I/O) pins that can be used to implement the sensors for the position of the boom and mass, or the skipper controls. To address this issue, a custom PCB was created, identical to the 9DOF Razor, which incorporates 18 I/O pins to supply power and allow various sensors to be connected.

To perform the task of creating the circuit, the team obtained the circuit schematics of the 9DOF Razor and created a new circuit schematic, found in Appendix I, using the same components with an addition of the 18 I/O pins. The program used to create the schematic is CadSoft EAGLE PCB Design Software, where the user can also create a PCB layout from the schematic. Creating this layout involved placing the components on a 3 x 2.5 inch board and tracing the components for connection through copper lines. Each component was chosen to be either through hole or surface mount technology (SMT) to minimize board size and easily create the connection traces. For connection between through hole and SMT components, the designer must add vias, a hole that is lined with metal to allow connection from the top layer of the board to the bottom layer. Using the steps provided above, the team created a two layered circuit board to order, as seen in Figure 23 below. The layout of the PCB can be found in Appendix J. The company the team used to create this board was Advanced Circuits, contacted through 4pcb.com.

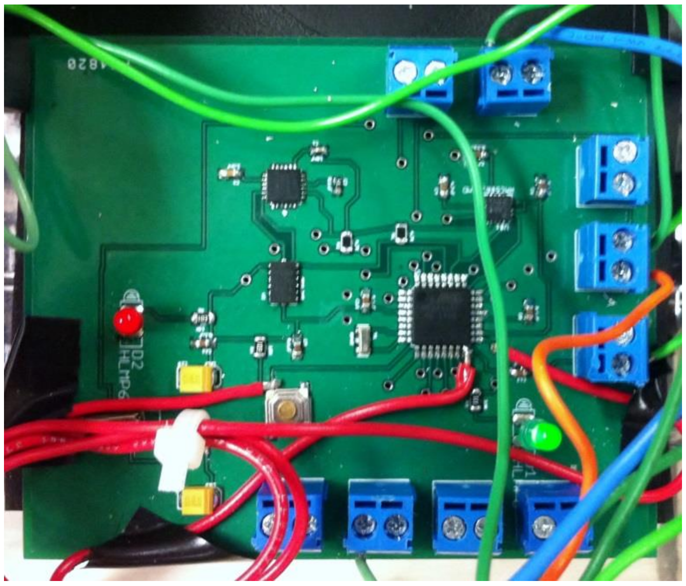


Figure 23: Two Layered PCB

Having the PCB complete, the team organized a parts list which can be found in Appendix K. Parts were ordered from Mouser Electronics, DigiKey Electronics, and Sparkfun. Some components, such as capacitors and resistors, are extremely small in size, an upwards of 0.6 x 0.3 mm, where a microscope was necessary to solder these components to the board. During the soldering process, flux was used to help the solder flow from the soldering iron to the components. Once all soldering was complete, the team used flux remover to clean the PCB and solder joints. Once the PCB was completely populated with components, the next step was to burn a bootloader on the Atmel microprocessor (Atmega328p) to use the Arduino IDE Software to implement the team's program code onto the PCB. This can be done on the Arduino IDE with an In-System Programming (ISP) connection to USB cable for communication to a computer. The ISP uses the MOSI, MISO, RESET, VCC, GND, and SCK pins for the bootloader; however, once burned, the pins can be used as I/O if needed. The PCB was placed inside a Pelican 1050 watertight case for protection in a wet environment as shown in Figure 24.

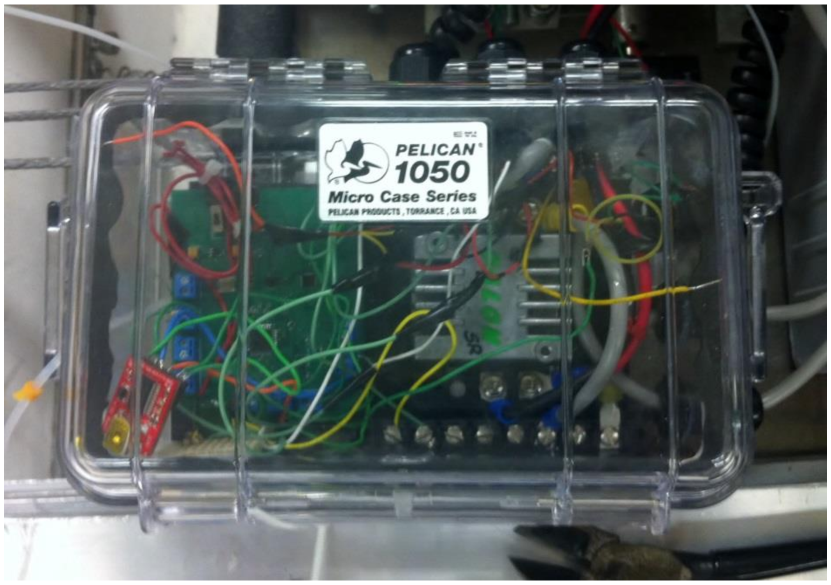


Figure 24: Watertight Pelican 1050 case

4.4.2 Sensor Calibration

With the Razor AHRS software we can calibrate the IMU to better position the sensor to resemble the boats movement. This data can be found in the Appendix M and the calibration was done through the help from the SparkFun electronics guide (Bartz, 2013). This code was implemented and modeled using Processing 2.1, a programming language and development environment, which was used to simulate the orientation of the IMU. Figure 18 shown below shows the GUI based environment used by Processing 2.1.

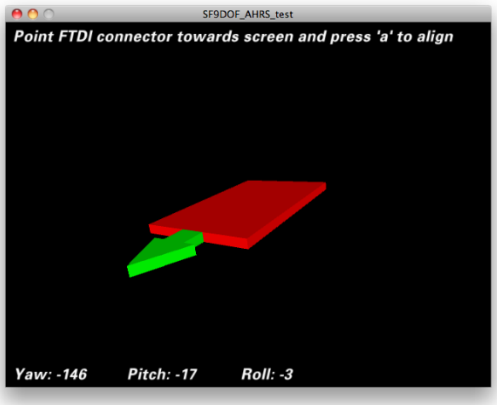


Figure 25: Computer Simulation of Gyroscope (Bartz, 2013)

4.4.3 Position of Mass

In measuring the position of the mass on the track, a 10K ohm 10 turn wire wound potentiometer, shown below in Figure 26, is used. There are several reasons the team chose to use a 10 turn potentiometer. First, the length of the track is approximately 9.2 turns with the gear ratio provided by the gear box. Second, the microprocessor takes a voltage input using a 5V reference to the potentiometer and correlates this voltage level with where the mass is positioned. The potentiometer is encased in a 3D printed housing and surrounded by silicon to prevent water from damaging this component.

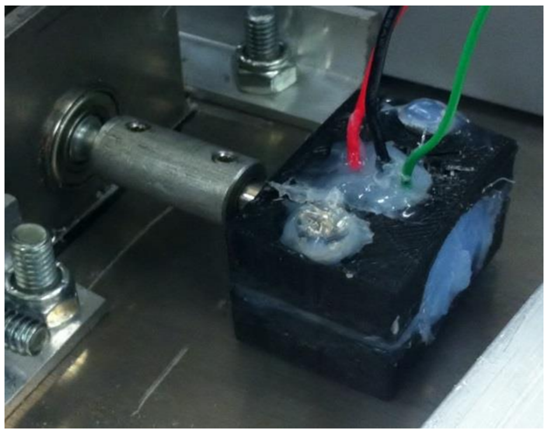


Figure 26: Mass Position Sensor

pinMode(MASSPOS_PIN, INPUT);

```
int PVAL;
PVAL = analogRead(MASSPOS_PIN);     //Read in potentiometer for location of the mass on the track
int pval = map(PVAL, 0, 1023, -50, 50);
```

Shown above is the code used to read the mass position sensor. Using the Arduino software, one can easily read the data from the potentiometer and constrain these values from -50 to 50. These values where chosen because the track measures to approximately 5 ft. and the team decided to have the corresponding mapping intervals of ten from the center to the end of the track on one end. This would therefore improve measuring accuracy and allow each increment to be one tenth of a foot from the center position to the edge of the track.

4.4.4 Position of Boom

To indicate whether the boom is port or starboard, a toggle switch, shown in Figure 27, was implemented. Using bungee cord and hooks, the toggle switch is triggered when the boom swings in either direction. This is critical to measure when the skipper is using the heel set dial from the skipper control panel.

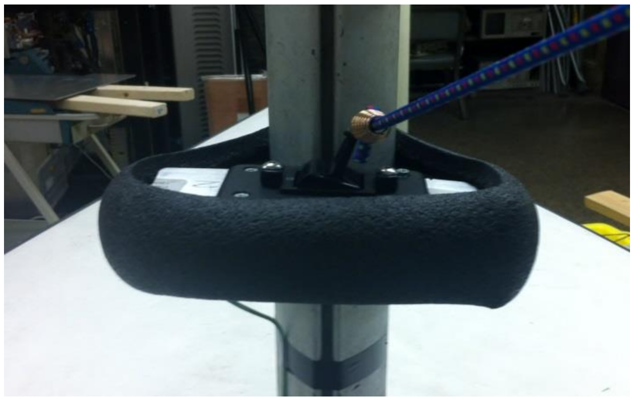


Figure 27: Boom Position Sensor

Source Code is as follows:

```
boomval = analogRead(BOOM_PIN);

if (boomval == 1){
   hval = -abs(hvalset);
}
else{
   hval = abs(hvalset);
}
```

As shown in the code above, notice how basic the implementation is because this boom position simply functions in negating the heel set value. This is important because the skipper needs the heel of the boat to be opposite to that of the boom's position. In Figure 27, notice how the toggle switch moves back and forth as a bungee cord creates the pull necessary to switch states. The bungee cord proved to be very successful because it could have some give from stretching therefore creating some hysteresis. This form of hysteresis allowed the toggling to occur after the boom has moved to a far enough position and eliminated the scenario of what to do if the boom was in a center position or within the a range at the center.

4.4.5 End of Track Sensors

In sensing the end of the track, the team decided to not implement push button bumpers or magnetic sensors; however, the team used the mass position sensor to determine the boundaries for each end of the track system. Using the microprocessor, the team implemented these boundaries in code shown below.

```
if ( pval < - 45 || pval > 45 || sensVal < -30 || sensVal > 30  ) //end of track & max heel check
   myservo.write(90+x);   //moves the motor in the opposite direction
```

The segment of code on the previous page checks for two conditions, the first being the position of the mass on the track, the second being the measured heel angle limits. The “pval” shown above refers to the value of the mass position sensor and the “sensVal” refers to the measured heel angle of the boat. The code checks to see whether or not the position of the boat is beyond the set limits which are -45 and 45. The actual max is -50 and 50 for the potentiometer mapping of extremes but for safety, we only go to -45 and 45 since we never want to reach the max limit of the potentiometer. These values are chosen because since the track length is approximately 5 ft., the track from center to end can be broken into 10 sections where one section is one tenth of a foot. Additionally, the code above moves the motor in the opposite direction of its current traverse if the measured heel angle goes above the skippers set heel angle. The x value in “myservo.write()” is the PID output which is being added to that center point at 90 for the servo motor Arduino controls which go from 0 to 180 where 90 is the break.

4.4.6 Skipper Controls

The skipper controls include an auto-center button and heel set dial that are within reach of the skipper. The auto-center button is a push button switch that when pressed, moves the weight to the middle of the boat. The heel set dial is a 100 ohm one turn potentiometer that sets the angle of heel between zero and fifteen degrees. The skipper control panels, shown below in Figure 28 (left), uses a cat5 cable to wire the potentiometer and switch to an electrical box positioned on the mast. This watertight electrical box, shown in Figure 28 (right), is used to make a connection between the cat5 cable and a spiral bound telephone wire. As depicted in Figure 29, the reason for this is to allow ample movement of the weigh platform and electronics without having issues of tightening or breaking cables. The code involved in the skipper control panel is shown below.

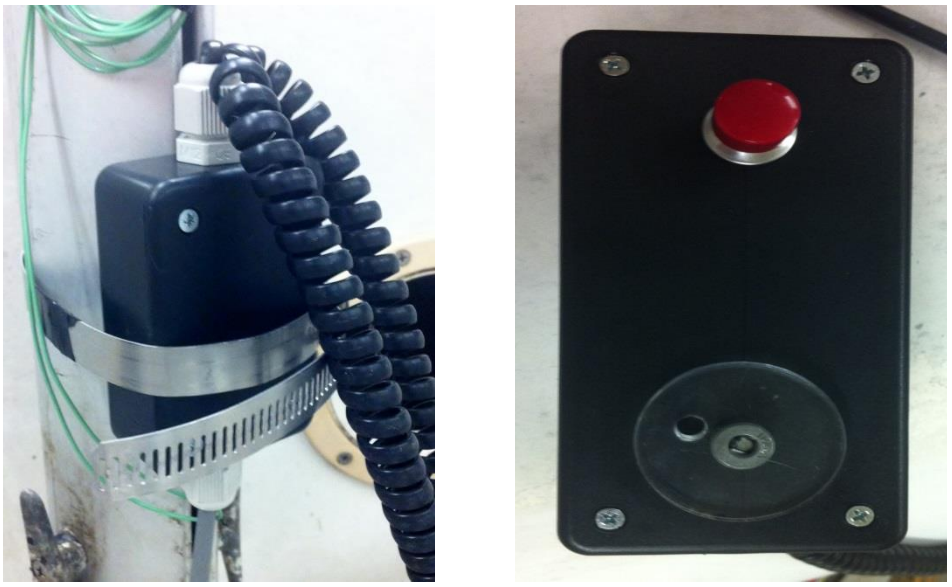


Figure 28: Watertight Electrical Box (left), Skipper Control Panel (right)

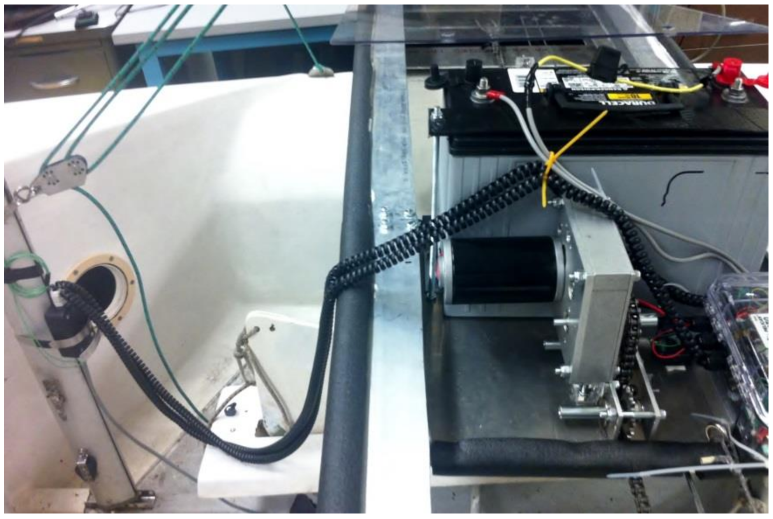


Figure 29: Spiral Wire for Boom Sensor and Skipper Controls

```
int heelset = analogRead(HEELSET_PIN);      //Reads in the heel set dial
int hvalset = map(heelset, 0, 1023, -20, 20);  //Constrains the limits from -20 to 20
```

```
if (autoc == HIGH){        //checks to see if the auto center button is high
  if(pval==0)
     myservo.write(90);   //stops the motor
  else
    myservo.write(90+y);  //centers the motor based on the mass position PID
}
else {
 if ( pval < - 30 || pval > 30 || sensVal < -30 || sensVal > 30  ) //end of track & max heel check
    myservo.write(90+x);   //moves the motor in the opposite direction
 else
    myservo.write(90-x);   //centers the motor at the heel set angle based on the measured heel PID
}
```

Upon initialization, the skipper sets the heel angle the system reads in this value. Notice how in the “hvalset” we used the map function to set a range for the heel set which we set from -20 to 20 degrees but in reality and for testing purposes this is limited to -15 and 15 degrees. This code checks whether or not the auto center button has been pressed and either centers the mass to position zero then stops or stabilizes the boat at the skippers set heel angle from the dial. As mentioned in the previous section on the about the end of track sensor, one can notice how it is now implemented as part of the skipper controls shown in the code above.

4.5 System Block Diagram

The system block diagram, shown below in Figure 30, illustrates the connections between the input sensors, input controls, and the output signal to drive the motor. First, our 12V power supply drives the talon motor controller as well as the 3.3V regulator, data sheet in Appendix N, to operate the microprocessor. The boom position sensor, mass position sensor, and IMU are all connected to the microprocessor, which interprets this information to drive the motor controller. The heel set dial and auto center button also connect to the microprocessor; however, the microprocessor will not perform any functions in accordance with these inputs unless the skipper manually uses them. As the microprocessor outputs a velocity value between 0 and 180, where the value of 90 corresponds to braking the motor, the motor controller, shown in Appendix O, outputs a PWM signal to create a duty cycle that correlates to that velocity value. The values of 0-89 move the mass starboard while the values of 91-180 move the mass to the port side. Having a value of 0 or 180 means the mass must move port or starboard as quickly as possible to counteract the forces acting on the boat.

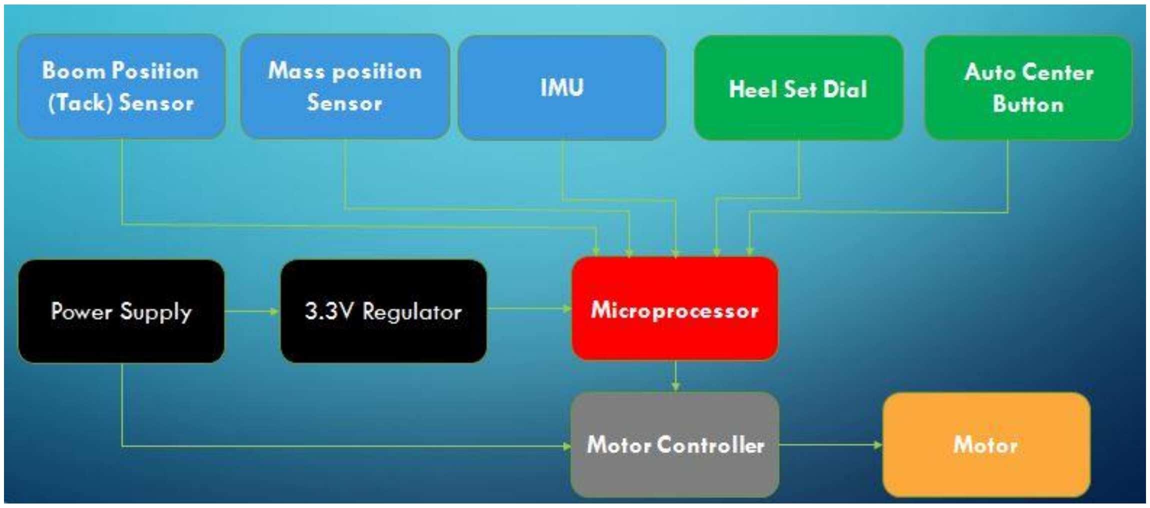


Figure 30: System Block Diagram

4.5 Stabilization Loop

An important aspect in combining all the different sensors and mechanical systems together is through the controls which can be seen in Appendix P. This portion of the document explains how all the sensors come together then feed into the microcontroller, PID, motor controller and ultimately driving the motor on the track. As you can see from the flow chart in Appendix E the system for this design needs to be robust and smooth since the team is dealing with unstable conditions on the water. These unstable conditions create low frequency noise to the system therefore there must be some sort of damping to the IMU sensor’s output which will go to out PID.

With the small signal noise calculated, the system can begin its process of damping the noise for better stability measurements. This can be done through the implementation of a lowpass filter in two ways; through the Arduino library’s own filtering tools or through hard coding of a Kalman filter. Smooth is a simple digital low-pass filter that is useful for smoothing sensor jitter or creating a delayed response to fast moving data. It uses a buffer variable and limits the amount of new data that reaches the output each time through the loop. Old data is used to make up the difference so that the response of the filter is slowed down (Badger, 2007). The Kalman filter is useful because it is not a filter in terms of frequencies, but an optimal estimator. This means that since it is recursive, new measurements can be processed as they arrive. This greatly minimizes Gaussian noise which is beneficial to eliminating the jitters from the boat merely sitting in the water. In the code in the next page you can see how the Kalman filter is set up and how the Arduino uses the smoothing function.

Arduino Smoothing Implementation Prototype

```
int smooth(int data, float filterVal, float smoothedVal){

  if (filterVal > 1){      // check to make sure param's are within range
    filterVal = .99;
  }
  else if (filterVal <= 0){
    filterVal = 0;
  }

  //feeds back the output into the input
  smoothedVal = (data * (1 - filterVal)) + (smoothedVal  *  filterVal);

  return (int)smoothedVal;
}
```

Kalman Filter Implementation Prototype

```
  float Q_angle  =  0.001; //0.001
  float Q_gyro   =  0.003;  //0.003
  float R_angle  =  0.03;  //0.03

  float x_angle = 0;
  float x_bias = 0;
  float P_00 = 0, P_01 = 0, P_10 = 0, P_11 = 0;
  float dt, y, S;
  float K_0, K_1;

float kalmanCalculate(float newAngle, float newRate,int looptime) {
  dt = float(looptime)/1000;
  x_angle += dt * (newRate - x_bias);
  P_00 +=  - dt * (P_10 + P_01) + Q_angle * dt;
  P_01 +=  - dt * P_11;
  P_10 +=  - dt * P_11;
  P_11 +=  + Q_gyro * dt;

  y = newAngle - x_angle;
  S = P_00 + R_angle;
  K_0 = P_00 / S;
  K_1 = P_10 / S;

  x_angle +=  K_0 * y;
  x_bias  +=  K_1 * y;
  P_00 -= K_0 * P_00;
  P_01 -= K_0 * P_01;
  P_10 -= K_1 * P_00;
  P_11 -= K_1 * P_01;

  return x_angle;
}
```

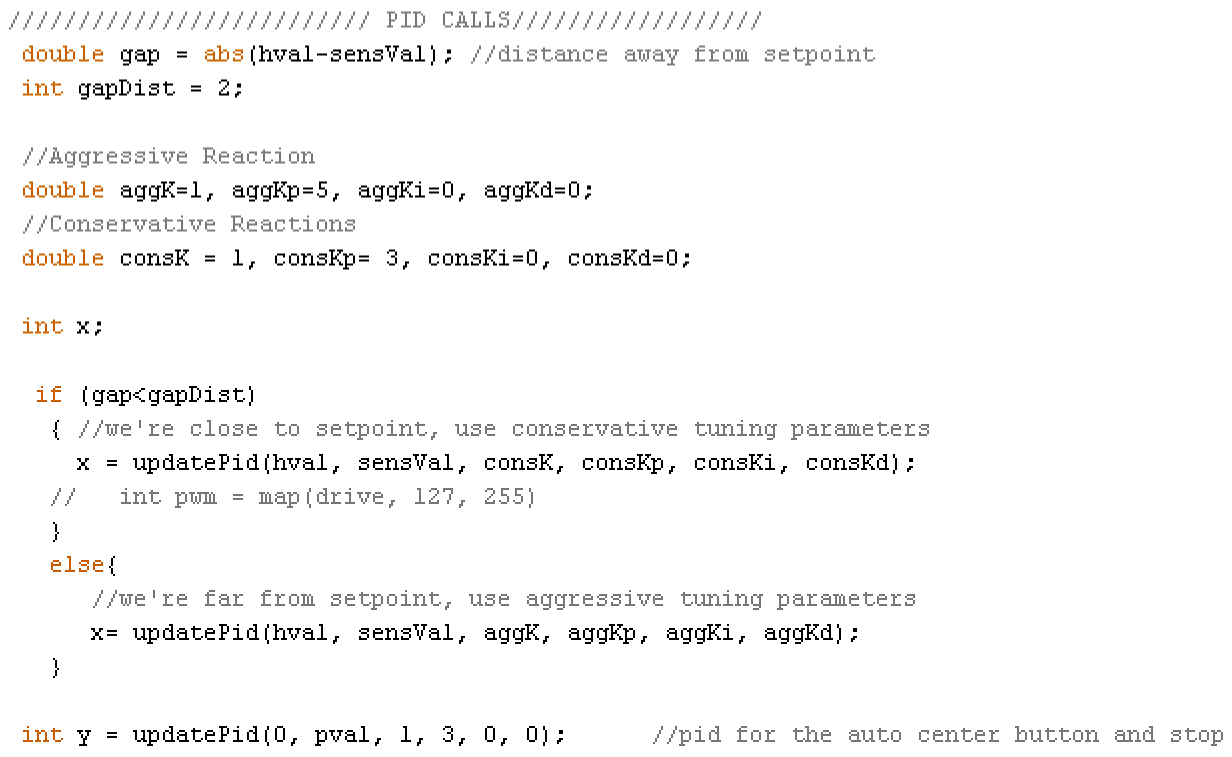
After filtering the measured angle of the boat, the system then needs to create a form of taking this measured value and comparing it to the skippers set heel value. This introduces the PID into the system which is extremely important to the control of how the motor reacts in stabilization. The code for the PID on the next page shows how the team set up the PID which then is called in the main loop. The function prototype is called “updatePid” which takes in the target position, current position, a gain K, Kp, Ki, and Kd. This implementation requires multiple trial and errors because the PID tuning process is quite tedious.

In the next bit of code from the main loop “updatePid” is called for the heel set stabilization and also for the auto center control. The “if” statement controls how aggressive the motor should move with respect to a margin of error, which in this case can be seen as 2 degrees. In stabilizing the boat through the heel set dial “x’ is represented as the PID output dealing with angles. The “y” integer is used when the auto center button is pressed where the target position is 0, “pval” is the measured position, and 1,3,0,0 refer to the gain (K) and Kp, Ki, Kd constants for the PID.

PID Prototype

```
unsigned long lastTime;
int last_error = 0;
int integrated_error = 0;
int pTerm = 0, iTerm = 0, dTerm = 0;
int error;

int updatePid(int targetPosition, int currentPosition, float K, int Kp, int Ki, int Kd) {

  /*How long since we last calculated*/
   unsigned long now = millis();
   double timeChange = (double)(now - lastTime);

  error = targetPosition - currentPosition;

  integrated_error += (error*timeChange);

  if (abs(error) < 2)    //creates some damping if error is insignificant
            error = 0;

  pTerm = Kp * error;
  iTerm = Ki * integrated_error;
  dTerm = Kd * (error - last_error)/timeChange;

  last_error = error;
  lastTime = now;

  return  constrain(K*(pTerm + iTerm + dTerm), -100, 100 );

}
```

PID Call in main loop()



```
///////////////////////////// PID CALLS///////////////////
 double gap = abs(hval-sensVal); //distance away from setpoint
 int gapDist = 2;

 //Aggressive Reaction
 double aggK=1, aggKp=5, aggKi=0, aggKd=0;
 //Conservative Reactions
 double consK = 1, consKp= 3, consKi=0, consKd=0;

 int x;

  if (gap<gapDist)
   { //we're close to setpoint, use conservative tuning parameters
     x = updatePid(hval, sensVal, consK, consKp, consKi, consKd);
   //   int pwm = map(drive, 127, 255)
   }
   else{
      //we're far from setpoint, use aggressive tuning parameters
      x= updatePid(hval, sensVal, aggK, aggKp, aggKi, aggKd);
   }

 int y = updatePid(0, pval, 1, 3, 0, 0);      //pid for the auto center button and stop
```

As previously described, the code above shows how we used the PID prototype to input our tuning parameters. These values can change based on the conditions on the water for either an aggressive or more conservative motor. Loop control plays a major role to this design and overall the concept of a mechanical system for the purpose of stabilization was proved to be achievable.

5.0 Discussion

In this section, the team performs an evaluation of the project, including how well the mechanical and electrical systems operate. The team also offers recommendations for future work on the project as well as several obstacles overcame while manufacturing and testing the product.

5.1 Mechanical

Several aspects of the mechanical design were altered during the build process due to factors such as availability of resources, cost limitations, or time limitations. One of the first modifications necessary was the reduction from two bearings per bracket to only one bearing per bracket. It was calculated that one bearing could support the force of the mass at full extension with 3G's of down force, which was calculated to be less than 225 pounds on a single bearing. The bearings selected had a working radial load of 300 pounds which would work for our application. In order to reduce cost, only one bearing was implemented in each bracket and preliminary tests confirmed that these brackets could withstand the forces applied from the track system without fault. The brackets that housed the bearings were also adjusted from the initial design. In order to allow for smooth transitions of the track the brackets were ground down to create a ramp up towards the bearing. Another design aspect that changed during the build process was the method of mounting the track to the boat. The initial design of an aluminum block milled to the contour of the boat's rail proved too heavy and costly to implement, so a lighter and more cost effective solution was devised. The simple solution involved mounting a length of PVC pipe to a screw-clamp mechanism that would fit underneath the rail of the boat and be able to firmly clamp the track to the boat. The two main advantages of this system are that it is able to fit securely under the rail and it could bend to the contours of the boat.

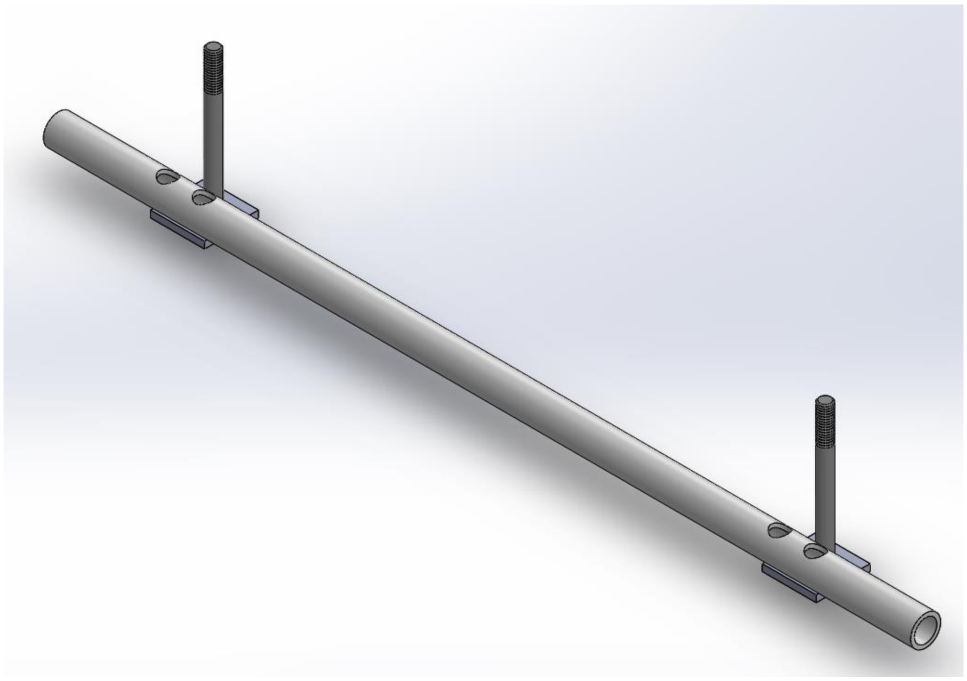


Figure 31: Mounting Clamp

It was also found that an active braking system was not necessary to implement. The gear ratio of the gearbox combined with the torque of the motor were sufficient to hold the mass at a specific position at any heel angle.

In addition, some aspects of the final design were overlooked during the design process, but were easily implemented during the build process. Once such addition was the foam spacers mounted underneath each bearing bracket. These allow for the elevation of the bearing to be adjusted in order to remove any gaps between the bearings and tracks that may have occurred during machining. Also, tapering of the ends of the tracks was necessary to aid in the smooth transition of the track on and off of the bearings. Finally, it was necessary to add foam padding to many of the sharp edges of the track in order to reduce risk of the skipper injuring him/herself should they be thrown toward the device in rough waters.

5.2 Electrical

The PCB for this project functions identical to the 9DOF Razor used for initial testing purposes with the addition of adding more sensors to perform the tasks required. While testing the PCB, the team noticed that the voltage regular provided 3.3V to the entire board. All traces were complete and the PCB provided full functionality. During the implementation of the mass position sensor, the potentiometer was defective. Once this component was replaced, the project sensed the position of the mass as expected. Upon first creating the boom position sensor with a small toggle switch, the team decided to use a different toggle switch that can implement using a bungee cord which is hooked to the boom and the toggle switch, where the boom has six inches of wiggle room when directly in the center of the boat to toggle the switch. This proved to be a better solution for our project to give the mass addition time to adjust and stabilize the boat while the boom swings from port to Starboard. Additionally, the code for this project performed all the necessary tasks our goals aimed towards. Stabilization at a desired heel was achieved, an auto center feature was implemented, the device senses whether the boom is port or starboard, and different traversal speeds are achieved for different scenarios.

5.3 Challenges

One challenge the team overcame was to solder such small devices onto the PCB board. With the help of Robert Boisse of the ECE Department, who soldered the magnetometer, accelerometer, gyroscope, and microprocessor, the team was able to populate the circuit board. Along with soldering, the team was unaware of the process to burn a bootloader onto a microprocessor chip. The current design of the PCB had not incorporated an ISP connection to easily perform such a task. The team proceeded to solder additional wires where necessary to burn the bootloader using the AVR ISP to USB board and the help of Joseph St. Germain. A recommendation for anyone creating a custom PCB which uses a microprocessor is to add an ISP connection to implement burning a bootloader in an easy fashion.

The second challenge dealt with feedback control and speeds for the motor moving across the track. As much as the skipper would want the motor to traverse as fast as possible, there needed to be some form of slowing the motor down as it got to its expected position. Without slowing down, the motors momentum would quickly jerk the boat as it abruptly stopped at its destination. This is not practical when needing a boat to properly remain stable, therefore, a solution was found by linking the PID response to the motor controller’s speed control.

This project was very metal work intensive, and the group ran into several problems trying to get access to the right tools. One of the problems was the difficulty of getting into the on campus machine shops. Each time the members went in to try to use a machine they were pushed off onto another employee a few days later, who then pushed us off to another employee in a seemingly endless cycle. Fortunately, Robert Boisse allowed the use of machines in his lab to get a lot of the metal working done. However, the most difficult metal cut could not be done one his band saw, so the team had to use a metal grinder and a straight board to cut the quarter inch rectangular tubing in half, a timely process.

5.4 Recommendations

This project has the potential for several follow-on projects in order to create a fully autonomous sailing vessel. It is also possible for the project teams to continue or work on the current stabilization system. Some work that could be done on the project is an adjustment of the PID loop and current code to potentially provide faster response times. The weight of the mechanical system can also be reduced by perforating the aluminum channels. This would allow additional weight to be added to the plate in order to increase righting moment while staying under the maximum weight requirement of 150 pounds.

Add on projects include the development of an auto-trimming device for controlling the trim of the gybe. This can be accomplished by measuring the air flow over both sides of the sail and trimming the sail to the ideal wind currents. Another possible extension to the project would be to add additional weight to the current system and to develop a robotic apparatus that can control the rudder, ultimately controlling the steering. This would be able to create the righting force of the skipper as well as drive the boat, the other main role of the skipper.

6.0 Social Implications

In developing a robotic system to be used in a confined and user compatible environment, it is important to consider the societal impact. This includes looking into safety measures going along with creating this device, as well as the implications of introducing an automated machine to a recreational sport.

The first main concern with this device is that the user will be in very close quarters to this robot and raises the potential for the device to harm them or to act in a manner that could cause the user to be harmed. Isaac Asimov devised three laws of robotics, which helped guide our initial concerns, and are listed below (Powerhouse Museum, 2002).

1. A robot may not injure a human being, or, through inaction, allow a human to come to harm
2. A robot must obey orders given to him by human beings except where such orders would conflict with the first law
3. A robot must protect its own existence, as long as such protection does not conflict with the first and second laws.

Looking at the first law, there are two main components, harm through action or harm through inaction. On the Flying Junior vessel, the user is placed in close proximity to the device, presenting need risk factors that needed to be addressed. Even without a robotic system the Flying Junior is dangerous and it is easy for the skipper to injure themselves when moving at high speeds around the boat. The project had to incorporate safety factors for even the static system, including covering all sharp edges and locations where the user could be cut. To accomplish this, padding was added to all exposed edges of the project so that the skipper would not be able to cut themselves on the sharp aluminum. In addition, the bolts used in the project were capped so that contact with them would not result in injury. Since the system is in an aquatic environment safety factors around electricity were addressed. All sensors were housed in water tight casing in order to avoid damage as well as the risk of electrocution. This was important because there are two sensors which the user comes in direct contact with. The second major concern with water and electricity was the battery. The battery that was selected has the ability to move as high speeds without releasing battery acid, which was essential for this application. A sealed absorbed glass-mat deep cycle battery was used, which has a sealed case and a spill proof design, both protecting the user as well as the environment. The second aspect of the battery is the risk of the battery being submerged and providing the skipper, who may be in the water, with an electrical shock. To address this rubber caps were placed on the top of the battery to create a water tight seal.

The second component of the first of Asimov's laws is the dangers of inaction. The system must be able to sense when it needs to move in order to adjust the heel of the boat. If the mass remained stationary when a large gust of wind came it is possible for the boat to flip without the needed righting force to hold it flat. In addition if the user began to lose control of the boat it is important for the system to have an emergency center button as to allow the skipper to right the boat if it does flip. This also allows the user to center the robot if they want to operate the system independently. It is important to incorporate user functionality as much as possible in this device. This is where the second law is used, where the robot must obey orders given to him by the human. By creating an override button the robot will center and allow the user to decide where to go from there.

There are social implications with integrating a robotic system into a sport competition with actions usually performed by humans. The main aspect of the crew that this system replaces is the crew's movement of weight which includes, sitting in the boat, leaning to windward or leeward, and hiking out. The latter of those options require the use of both leg and abdominal muscles, which overtime causes fatigue in the sailor. This system would not experience the buildup of lactic acid and, debatably, gives the system an unfair advantage over the competition. Although it does not have the ability to preemptively sense changes in the wind, it has the ability to hold a set heel angle very well. This gives it a performance advantage of human beings in some aspects. Robots like the TOPIO 3.0 robot, which has the ability to play and beat humans in Ping Pong, have been at the center of debate about integrating robotics into sports. A computer has the ability to process information at higher speeds than human beings giving it an advantage over the competition. Arguments have been made that robotic devices in sports would be unbeatable and are not fair competitors against human beings. Putting our system into a competition against humans would create societal problems and debates resisting the integration.

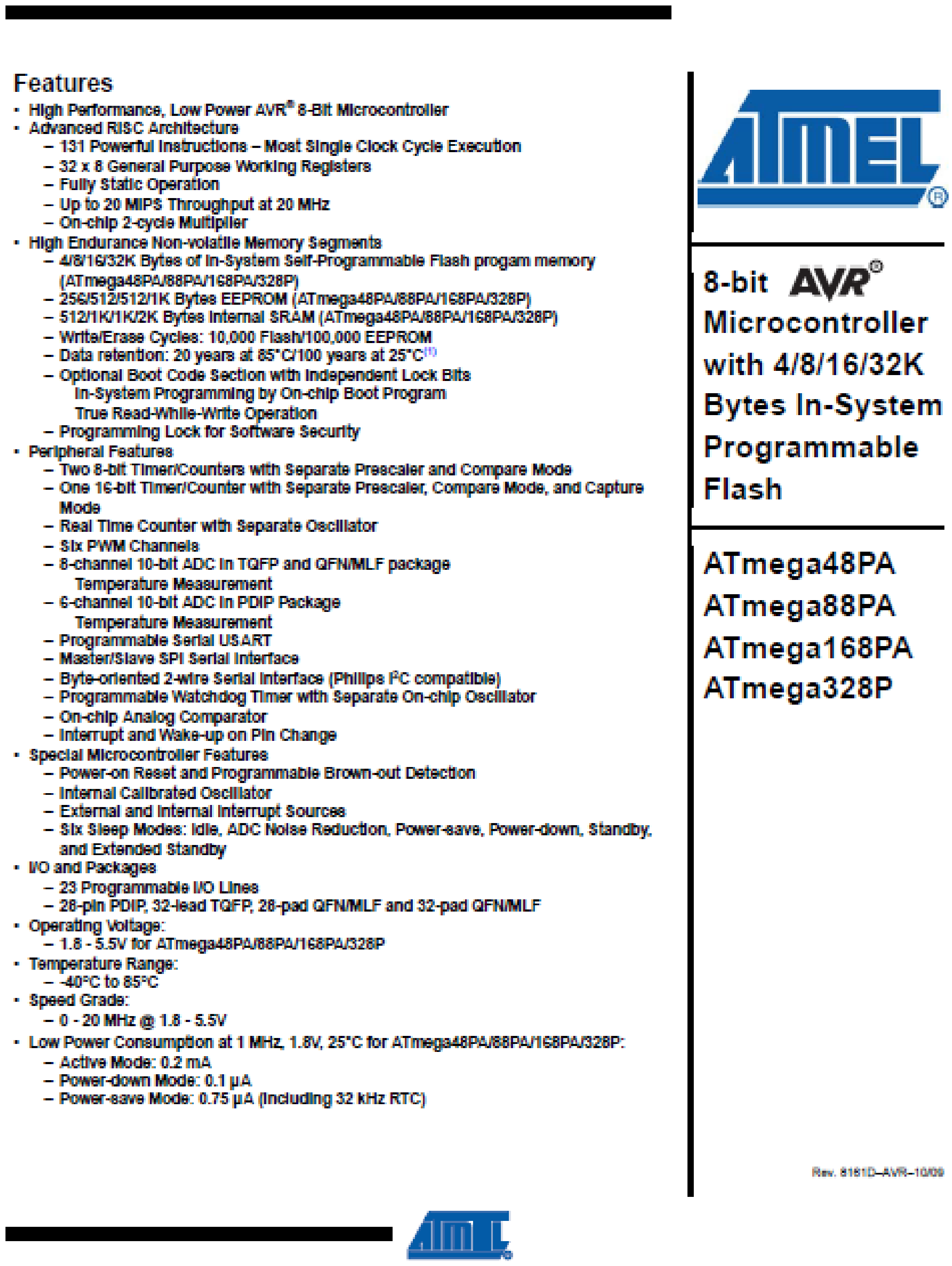
The robotic stabilization device has several social implications associated with it. It was important to address and resolve many of these issues such as user safety, environmental effects and public acceptance. The final stabilization system produced by the team incorporates safety factors, emergency prevention measures and overall user compatibility in order to provide a safe and functional robotic system.

7.0 Conclusion

In conclusion, the scope of this project focused on one particular aspect of the operation of this boat: the crew. By replicating this action with an autonomous device, the crew would be eliminated and it would then be possible to operate a Flying Junior solo. Using an IMU, the heel angle of the boat in the water can be measured. By setting a desired heel angle on a control panel, a mass driven by a motor is moved laterally along a track in order to adjust the heel angle accordingly. The goal of this project is to allow a sailor to safely operate a Flying Junior dinghy without a crew and be able to maintain full control of the boat. The team's objectives were to design a weighted mechanical system to replicate the righting force of a 150 lb. crew, adjust a moving mass along a track to correct the heel to the desired angle, create a device that weighs less than 150 lbs., and to provide enough power for multiple hours of continuous use. The end result of this project, based on ground simulation results, is a device that can be easily attached to any Flying Junior dingy, weighs less than an average human being, and can maintain a specific heel angle as well as an experienced crew is able to.

The device is a weighted track system made of 6061 aluminum alloy in a two rail elevator design that can extend in both direction, providing a righting force of 466 ft-lbs. Sealed ball bearings line the rails to allow for smooth travel of the rails and weight platform. The mechanism is driven by active and passive lines which allow the platform to travel at 4 ft. /sec. The electrical components include a CIM motor, Talon motor controller, deep cycle marine battery, and a custom PCB that replicates the function of SparkFun's 9DOF Razor. Various sensor inputs including potentiometers and switch are implemented to sense the position of the boom, the position of the mass, and a skipper control panel which allows the user to set a desired heel angle and center the mass. The total system weighed in at 130 lbs., less than that of an average crew. Refer to Appendix Q for the total cost of the project.

Appendix A



Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32K Bytes of In-System Self-Programmable Flash progam memory (ATmega48PA/88PA/168PA/328P)
  - 256/512/512/1K Bytes EEPROM (ATmega48PA/88PA/168PA/328P)
  - 512/1K/1K/2K Bytes Internal SRAM (ATmega48PA/88PA/168PA/328P)
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C[1]
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I^{2}C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V for ATmega48PA/88PA/168PA/328P
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 20 MHz @ 1.8 - 5.5V
- Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48PA/88PA/168PA/328P:
  - Active Mode: 0.2 mA
  - Power-down Mode: 0.1 µA
  - Power-save Mode: 0.75 µA (Including 32 kHz RTC)

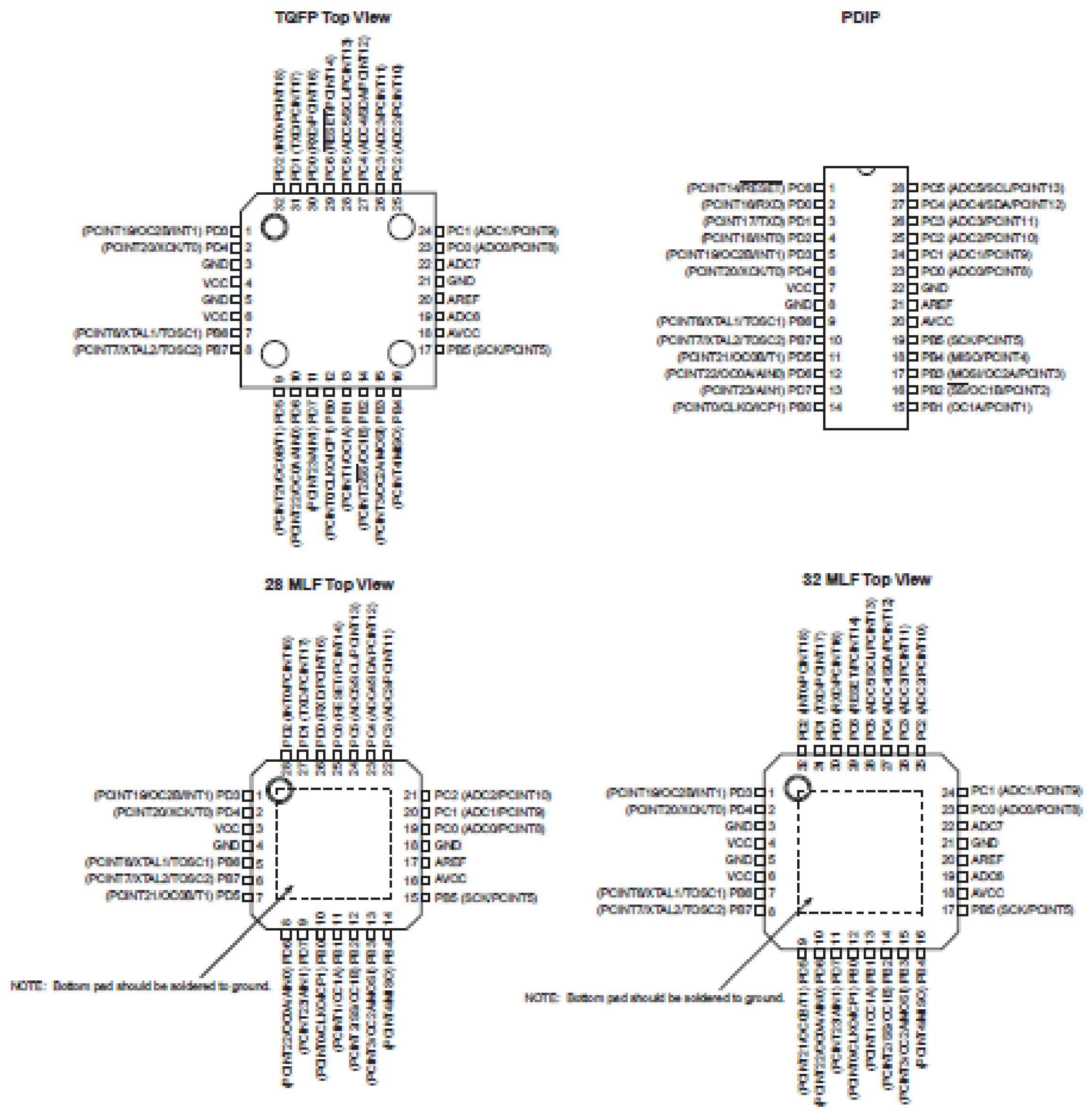
8-bit AVR® Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash

ATmega48PA
ATmega88PA
ATmega168PA
ATmega328P

Rev. 8161D–AVR–10/09

1. Pin Configurations

Figure 1-1. Pinout ATmega48PA/88PA/168PA/328P



1.1 Pin Descriptions

1.1.1 VCC

Digital supply voltage.

1.1.2 GND

Ground.

1.1.3 Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB7..6 is used as TOSC2..1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

The various special features of Port B are elaborated in "Alternate Functions of Port B" on page 82 and "System Clock and Clock Options" on page 26.

1.1.4 Port C (PC5:0)

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5..0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

1.1.5 PC6/$\overline{\text{RESET}}$

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. The minimum pulse length is given in Table 28-3 on page 318. Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated in "Alternate Functions of Port C" on page 85.

1.1.6 Port D (PD7:0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

The various special features of Port D are elaborated in "Alternate Functions of Port D" on page 88.

1.1.7 AV$_{CC}$

AV$_{CC}$ is the supply voltage pin for the A/D Converter, PC3:0, and ADC7:6. It should be externally connected to V$_{CC}$, even if the ADC is not used. If the ADC is used, it should be connected to V$_{CC}$ through a low-pass filter. Note that PC6..4 use digital supply voltage, V$_{CC}$.

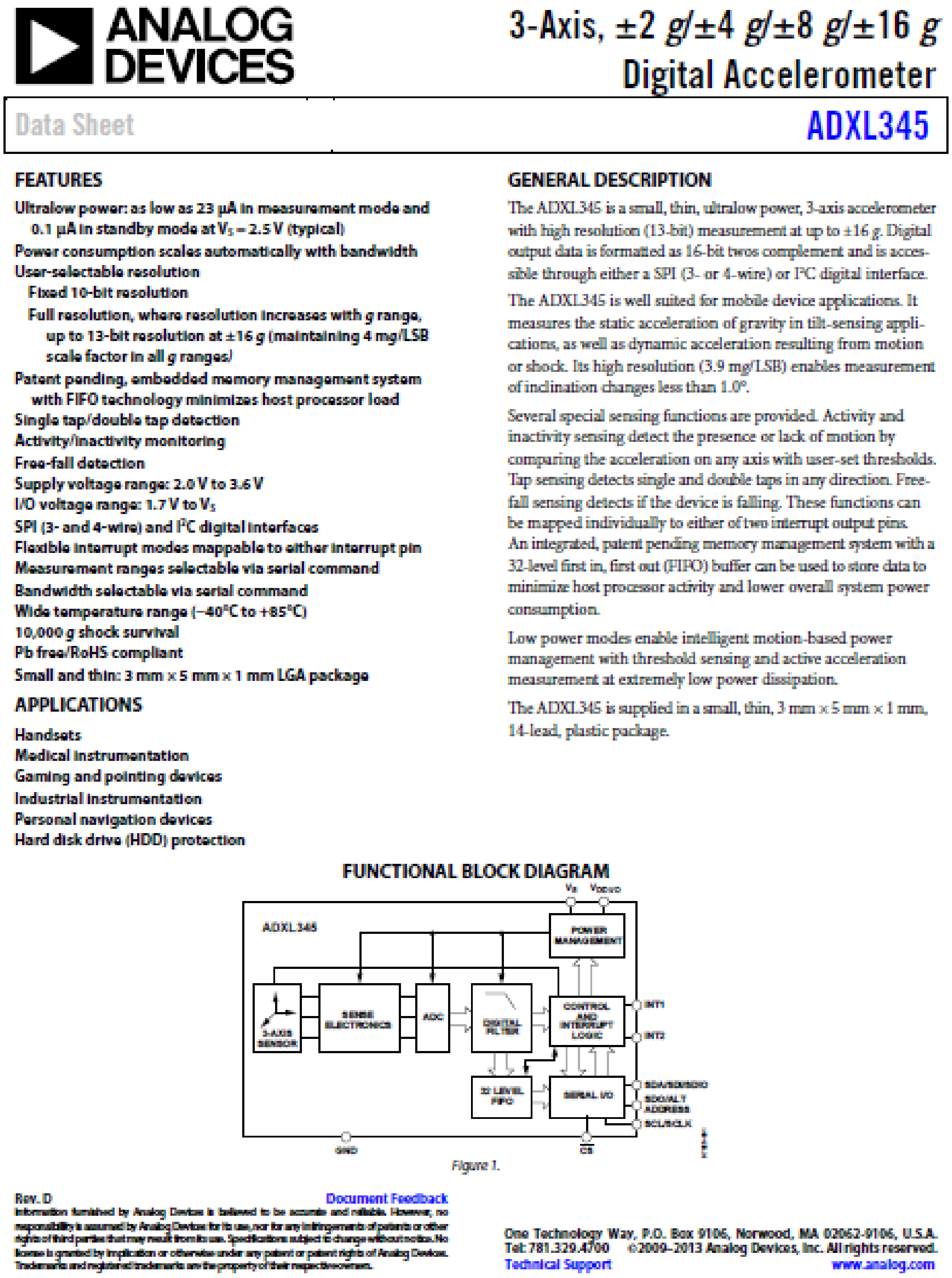
1.1.8 AREF

AREF is the analog reference pin for the A/D Converter.

1.1.9 ADC7:6 (TQFP and QFN/MLF Package Only)

In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

Appendix B



ANALOG DEVICES

3-Axis, ±2 *g*/±4 *g*/±8 *g*/±16 *g* Digital Accelerometer

Data Sheet **ADXL345**

FEATURES

- Ultralow power: as low as 23 μA in measurement mode and 0.1 μA in standby mode at V_S = 2.5 V (typical)
- Power consumption scales automatically with bandwidth
- User-selectable resolution
  - Fixed 10-bit resolution
  - Full resolution, where resolution increases with *g* range, up to 13-bit resolution at ±16 *g* (maintaining 4 m*g*/LSB scale factor in all *g* ranges)
- Patent pending, embedded memory management system with FIFO technology minimizes host processor load
- Single tap/double tap detection
- Activity/inactivity monitoring
- Free-fall detection
- Supply voltage range: 2.0 V to 3.6 V
- I/O voltage range: 1.7 V to V_S
- SPI (3- and 4-wire) and I^2C digital interfaces
- Flexible interrupt modes mappable to either interrupt pin
- Measurement ranges selectable via serial command
- Bandwidth selectable via serial command
- Wide temperature range (−40°C to +85°C)
- 10,000 *g* shock survival
- Pb free/RoHS compliant
- Small and thin: 3 mm × 5 mm × 1 mm LGA package

APPLICATIONS

- Handsets
- Medical instrumentation
- Gaming and pointing devices
- Industrial instrumentation
- Personal navigation devices
- Hard disk drive (HDD) protection

GENERAL DESCRIPTION

The ADXL345 is a small, thin, ultralow power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ±16 *g*. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3- or 4-wire) or I^2C digital interface.

The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (3.9 m*g*/LSB) enables measurement of inclination changes less than 1.0°.

Several special sensing functions are provided. Activity and inactivity sensing detect the presence or lack of motion by comparing the acceleration on any axis with user-set thresholds. Tap sensing detects single and double taps in any direction. Free-fall sensing detects if the device is falling. These functions can be mapped individually to either of two interrupt output pins. An integrated, patent pending memory management system with a 32-level first in, first out (FIFO) buffer can be used to store data to minimize host processor activity and lower overall system power consumption.

Low power modes enable intelligent motion-based power management with threshold sensing and active acceleration measurement at extremely low power dissipation.

The ADXL345 is supplied in a small, thin, 3 mm × 5 mm × 1 mm, 14-lead, plastic package.

FUNCTIONAL BLOCK DIAGRAM

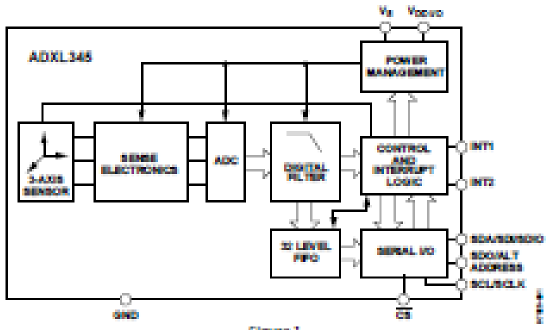
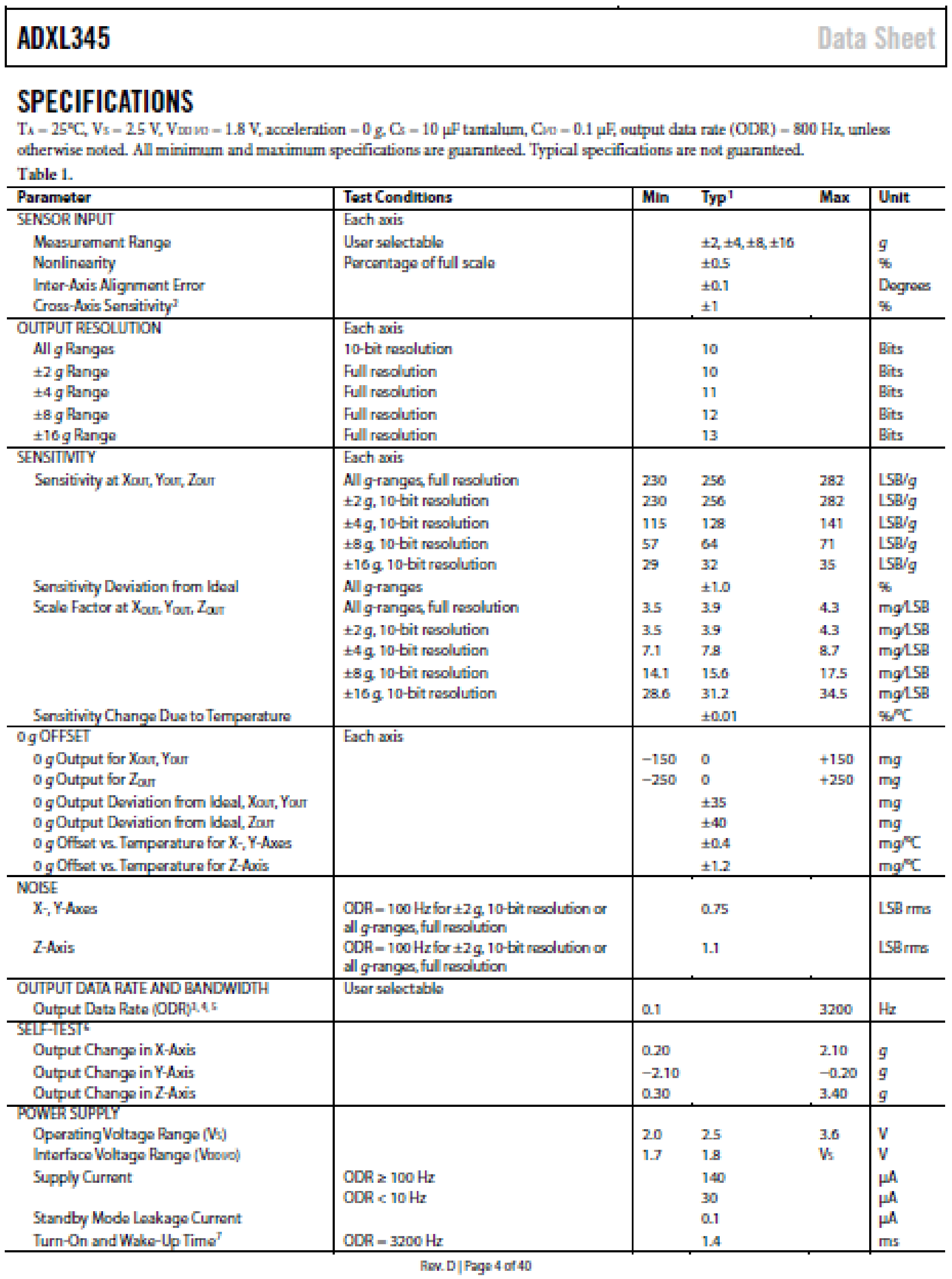


Figure 1.


ADXL345 Data Sheet

SPECIFICATIONS

T_A = 25°C, V_S = 2.5 V, $V_{DD I/O}$ = 1.8 V, acceleration = 0 *g*, C_S = 10 μF tantalum, $C_{I/O}$ = 0.1 μF, output data rate (ODR) = 800 Hz, unless otherwise noted. All minimum and maximum specifications are guaranteed. Typical specifications are not guaranteed.

Table 1.

Parameter	Test Conditions	Min	Typ[1]	Max	Unit
SENSOR INPUT	Each axis				
Measurement Range	User selectable		±2, ±4, ±8, ±16		*g*
Nonlinearity	Percentage of full scale		±0.5		%
Inter-Axis Alignment Error			±0.1		Degrees
Cross-Axis Sensitivity[2]			±1		%
OUTPUT RESOLUTION	Each axis				
All *g* Ranges	10-bit resolution		10		Bits
±2 *g* Range	Full resolution		10		Bits
±4 *g* Range	Full resolution		11		Bits
±8 *g* Range	Full resolution		12		Bits
±16 *g* Range	Full resolution		13		Bits
SENSITIVITY	Each axis				
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	All *g*-ranges, full resolution	230	256	282	LSB/*g*
	±2 *g*, 10-bit resolution	230	256	282	LSB/*g*
	±4 *g*, 10-bit resolution	115	128	141	LSB/*g*
	±8 *g*, 10-bit resolution	57	64	71	LSB/*g*
	±16 *g*, 10-bit resolution	29	32	35	LSB/*g*
Sensitivity Deviation from Ideal	All *g*-ranges		±1.0		%
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	All *g*-ranges, full resolution	3.5	3.9	4.3	m*g*/LSB
	±2 *g*, 10-bit resolution	3.5	3.9	4.3	m*g*/LSB
	±4 *g*, 10-bit resolution	7.1	7.8	8.7	m*g*/LSB
	±8 *g*, 10-bit resolution	14.1	15.6	17.5	m*g*/LSB
	±16 *g*, 10-bit resolution	28.6	31.2	34.5	m*g*/LSB
Sensitivity Change Due to Temperature			±0.01		%/°C
0 *g* OFFSET	Each axis				
0 *g* Output for X_{OUT}, Y_{OUT}		−150	0	+150	m*g*
0 *g* Output for Z_{OUT}		−250	0	+250	m*g*
0 *g* Output Deviation from Ideal, X_{OUT}, Y_{OUT}			±35		m*g*
0 *g* Output Deviation from Ideal, Z_{OUT}			±40		m*g*
0 *g* Offset vs. Temperature for X-, Y-Axes			±0.4		m*g*/°C
0 *g* Offset vs. Temperature for Z-Axis			±1.2		m*g*/°C
NOISE					
X-, Y-Axes	ODR = 100 Hz for ±2 *g*, 10-bit resolution or all *g*-ranges, full resolution		0.75		LSB rms
Z-Axis	ODR = 100 Hz for ±2 *g*, 10-bit resolution or all *g*-ranges, full resolution		1.1		LSB rms
OUTPUT DATA RATE AND BANDWIDTH	User selectable				
Output Data Rate (ODR)[3, 4, 5]		0.1		3200	Hz
SELF-TEST[6]					
Output Change in X-Axis		0.20		2.10	*g*
Output Change in Y-Axis		−2.10		−0.20	*g*
Output Change in Z-Axis		0.30		3.40	*g*
POWER SUPPLY					
Operating Voltage Range (V_S)		2.0	2.5	3.6	V
Interface Voltage Range ($V_{DD I/O}$)		1.7	1.8	V_S	V
Supply Current	ODR ≥ 100 Hz		140		μA
	ODR < 10 Hz		30		μA
Standby Mode Leakage Current			0.1		μA
Turn-On and Wake-Up Time[7]	ODR = 3200 Hz		1.4		ms

Rev. D | Page 4 of 40

PIN CONFIGURATION AND FUNCTION DESCRIPTIONS

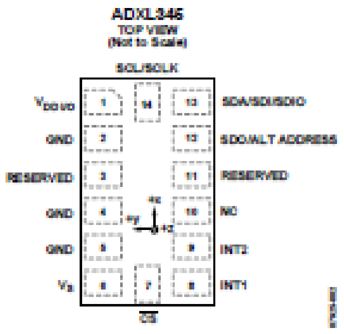
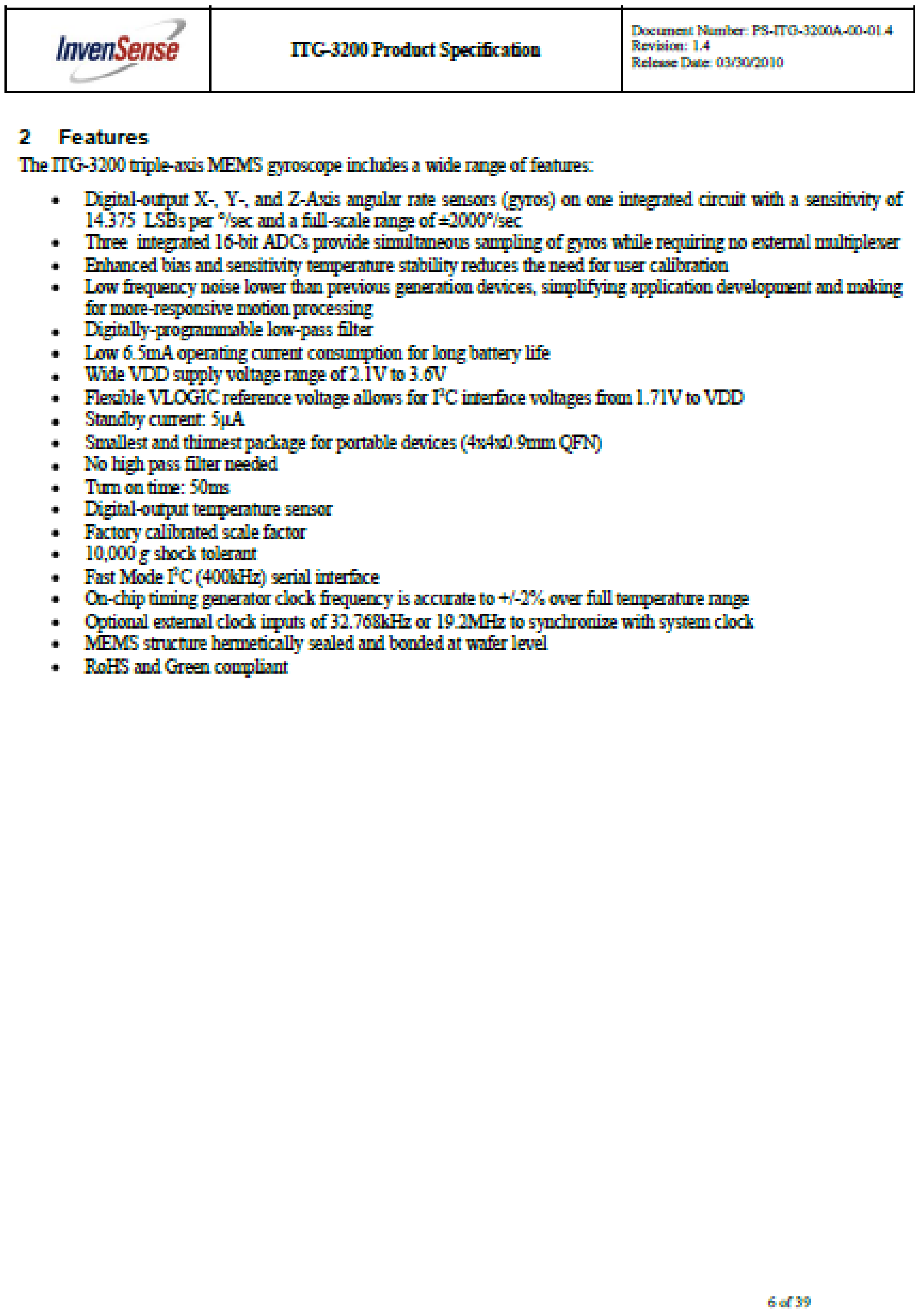


Figure 3. Pin Configuration (Top View)

Table 5. Pin Function Descriptions

Pin No.	Mnemonic	Description
1	$V_{DD\ I/O}$	Digital Interface Supply Voltage.
2	GND	This pin must be connected to ground.
3	RESERVED	Reserved. This pin must be connected to V_S or left open.
4	GND	This pin must be connected to ground.
5	GND	This pin must be connected to ground.
6	V_S	Supply Voltage.
7	$\overline{CS}$	Chip Select.
8	INT1	Interrupt 1 Output.
9	INT2	Interrupt 2 Output.
10	NC	Not Internally Connected.
11	RESERVED	Reserved. This pin must be connected to ground or left open.
12	SDO/ALT ADDRESS	Serial Data Output (SPI 4-Wire)/Alternate I²C Address Select (I²C).
13	SDA/SDI/SDIO	Serial Data (I^2C)/Serial Data Input (SPI 4-Wire)/Serial Data Input and Output (SPI 3-Wire).
14	SCL/SCLK	Serial Communications Clock. SCL is the clock for I²C, and SCLK is the clock for SPI.

Appendix C



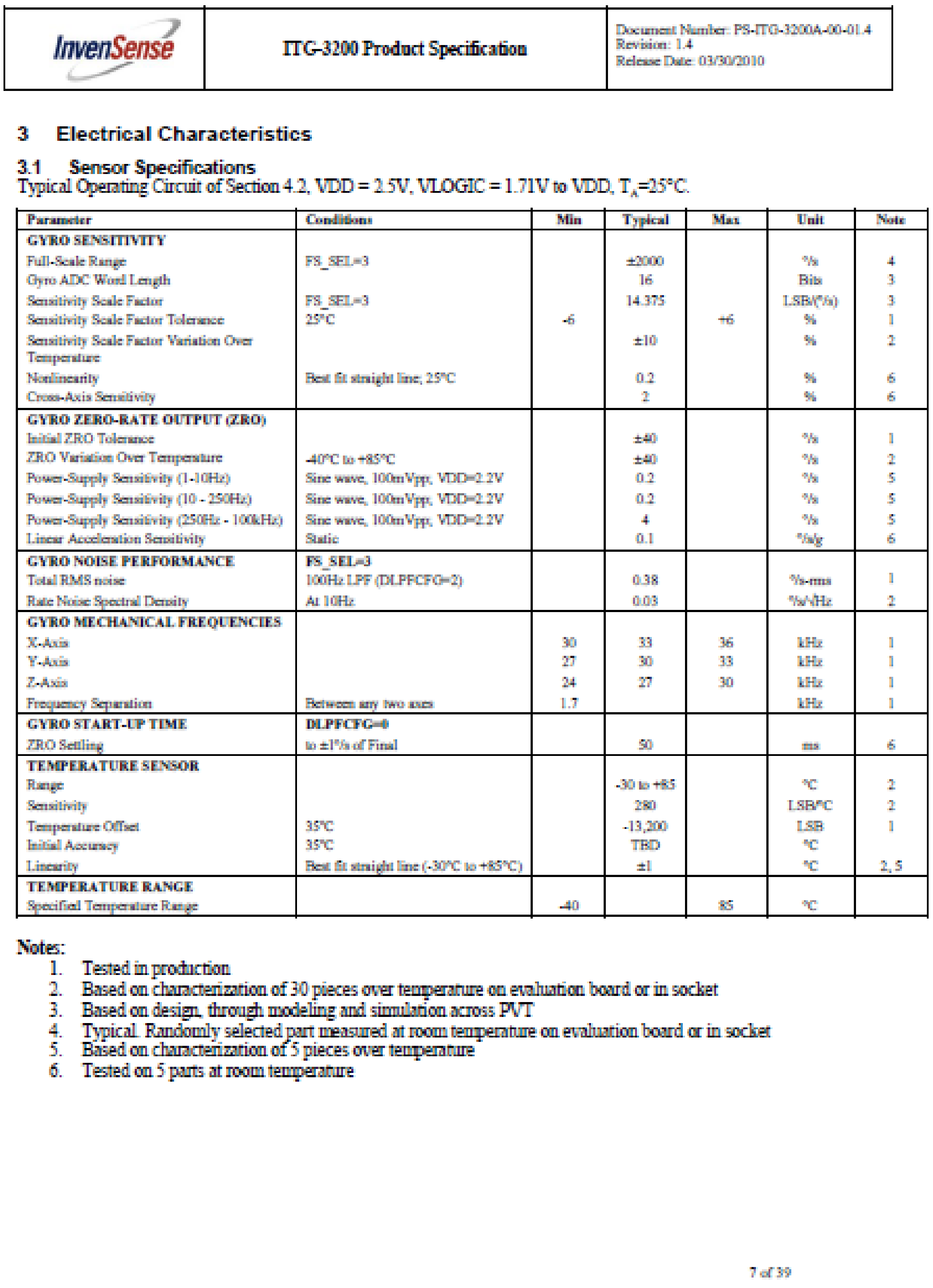
	ITG-3200 Product Specification	Document Number: PS-ITG-3200A-00-01.4 Revision: 1.4 Release Date: 03/30/2010

2 Features

The ITG-3200 triple-axis MEMS gyroscope includes a wide range of features:

- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyros) on one integrated circuit with a sensitivity of 14.375 LSBs per °/sec and a full-scale range of ±2000°/sec
- Three integrated 16-bit ADCs provide simultaneous sampling of gyros while requiring no external multiplexer
- Enhanced bias and sensitivity temperature stability reduces the need for user calibration
- Low frequency noise lower than previous generation devices, simplifying application development and making for more-responsive motion processing
- Digitally-programmable low-pass filter
- Low 6.5mA operating current consumption for long battery life
- Wide VDD supply voltage range of 2.1V to 3.6V
- Flexible VLOGIC reference voltage allows for I^{2}C interface voltages from 1.71V to VDD
- Standby current: 5µA
- Smallest and thinnest package for portable devices (4x4x0.9mm QFN)
- No high pass filter needed
- Turn on time: 50ms
- Digital-output temperature sensor
- Factory calibrated scale factor
- 10,000 *g* shock tolerant
- Fast Mode I^{2}C (400kHz) serial interface
- On-chip timing generator clock frequency is accurate to +/-2% over full temperature range
- Optional external clock inputs of 32.768kHz or 19.2MHz to synchronize with system clock
- MEMS structure hermetically sealed and bonded at wafer level
- RoHS and Green compliant

6 of 39



3 Electrical Characteristics

3.1 Sensor Specifications

Typical Operating Circuit of Section 4.2, VDD = 2.5V, VLOGIC = 1.71V to VDD, T_A=25°C.

Parameter	Conditions	Min	Typical	Max	Unit	Note
GYRO SENSITIVITY						
Full-Scale Range	FS_SEL=3		±2000		°/s	4
Gyro ADC Word Length			16		Bits	3
Sensitivity Scale Factor	FS_SEL=3		14.375		LSB/(°/s)	3
Sensitivity Scale Factor Tolerance	25°C	-6		+6	%	1
Sensitivity Scale Factor Variation Over Temperature			±10		%	2
Nonlinearity	Best fit straight line; 25°C		0.2		%	6
Cross-Axis Sensitivity			2		%	6
GYRO ZERO-RATE OUTPUT (ZRO)						
Initial ZRO Tolerance			±40		°/s	1
ZRO Variation Over Temperature	-40°C to +85°C		±40		°/s	2
Power-Supply Sensitivity (1-10Hz)	Sine wave, 100mVpp, VDD=2.2V		0.2		°/s	5
Power-Supply Sensitivity (10 - 250Hz)	Sine wave, 100mVpp, VDD=2.2V		0.2		°/s	5
Power-Supply Sensitivity (250Hz - 100kHz)	Sine wave, 100mVpp, VDD=2.2V		4		°/s	5
Linear Acceleration Sensitivity	Static		0.1		°/s/g	6
GYRO NOISE PERFORMANCE	**FS_SEL=3**					
Total RMS noise	100Hz LPF (DLPFCFG=2)		0.38		°/s-rms	1
Rate Noise Spectral Density	At 10Hz		0.03		°/s/√Hz	2
GYRO MECHANICAL FREQUENCIES						
X-Axis		30	33	36	kHz	1
Y-Axis		27	30	33	kHz	1
Z-Axis		24	27	30	kHz	1
Frequency Separation	Between any two axes	1.7			kHz	1
GYRO START-UP TIME	**DLPFCFG=0**					
ZRO Settling	to ±1°/s of Final		50		ms	6
TEMPERATURE SENSOR						
Range			-30 to +85		°C	2
Sensitivity			280		LSB/°C	2
Temperature Offset	35°C		-13,200		LSB	1
Initial Accuracy	35°C		TBD		°C	
Linearity	Best fit straight line (-30°C to +85°C)		±1		°C	2, 5
TEMPERATURE RANGE						
Specified Temperature Range		-40		85	°C	

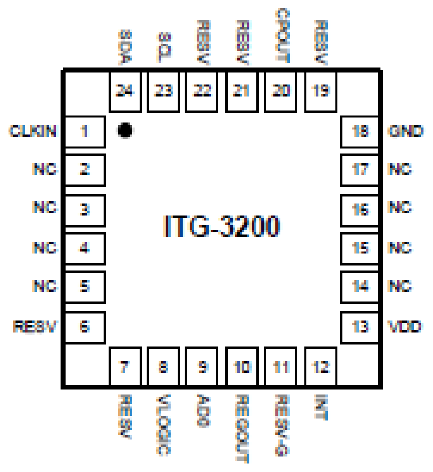
Notes:
1. Tested in production
2. Based on characterization of 30 pieces over temperature on evaluation board or in socket
3. Based on design, through modeling and simulation across PVT
4. Typical. Randomly selected part measured at room temperature on evaluation board or in socket
5. Based on characterization of 5 pieces over temperature
6. Tested on 5 parts at room temperature

4 Applications Information

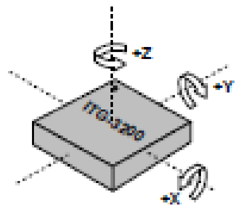
4.1 Pin Out and Signal Description

Number	Pin	Pin Description
1	CLKIN	Optional external reference clock input. Connect to GND if unused.
8	VLOGIC	Digital IO supply voltage. VLOGIC must be ≤ VDD at all times.
9	AD0	I²C Slave Address LSB
10	REGOUT	Regulator filter capacitor connection
12	INT	Interrupt digital output (totem pole or open-drain)
13	VDD	Power supply voltage
18	GND	Power supply ground
11	RESV-G	Reserved - Connect to ground.
6, 7, 19, 21, 22	RESV	Reserved. Do not connect.
20	CPOUT	Charge pump capacitor connection
23	SCL	I²C serial clock
24	SDA	I²C serial data
2, 3, 4, 5, 14, 15, 16, 17	NC	Not internally connected. May be used for PCB trace routing.

Top View



QFN Package
24-pin, 4mm x 4mm x 0.9mm



Orientation of Axes of Sensitivity
and Polarity of Rotation

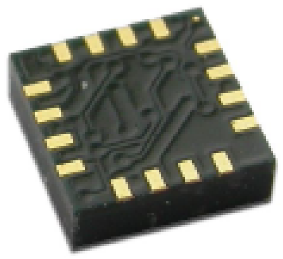
Appendix D

3-Axis Digital Compass IC HMC5883L

Honeywell

Advanced Information

The Honeywell HMC5883L is a surface-mount, multi-chip module designed for low-field magnetic sensing with a digital interface for applications such as low-cost compassing and magnetometry. The HMC5883L includes our state-of-the-art, high-resolution HMC118X series magneto-resistive sensors plus an ASIC containing amplification, automatic degaussing strap drivers, offset cancellation, and a 12-bit ADC that enables 1° to 2° compass heading accuracy. The I^{2}C serial bus allows for easy interface. The HMC5883L is a 3.0x3.0x0.9mm surface mount 16-pin leadless chip carrier (LCC). Applications for the HMC5883L include Mobile Phones, Netbooks, Consumer Electronics, Auto Navigation Systems, and Personal Navigation Devices.



The HMC5883L utilizes Honeywell's Anisotropic Magnetoresistive (AMR) technology that provides advantages over other magnetic sensor technologies. These anisotropic, directional sensors feature precision in-axis sensitivity and linearity. These sensors' solid-state construction with very low cross-axis sensitivity is designed to measure both the direction and the magnitude of Earth's magnetic fields, from milli-gauss to 8 gauss. Honeywell's Magnetic Sensors are among the most sensitive and reliable low-field sensors in the industry.

FEATURES	BENEFITS
▶ 3-Axis Magnetoresistive Sensors and ASIC in a 3.0x3.0x0.9mm LCC Surface Mount Package	▶ Small Size for Highly Integrated Products. Just Add a Micro-Controller Interface, Plus Two External SMT Capacitors Designed for High Volume, Cost Sensitive OEM Designs Easy to Assemble & Compatible with High Speed SMT Assembly
▶ 12-Bit ADC Coupled with Low Noise AMR Sensors Achieves 2 milli-gauss Field Resolution in ±8 Gauss Fields	▶ Enables 1° to 2° Degree Compass Heading Accuracy
▶ Built-In Self Test	▶ Enables Low-Cost Functionality Test after Assembly in Production
▶ Low Voltage Operations (2.16 to 3.6V) and Low Power Consumption (100 μA)	▶ Compatible for Battery Powered Applications
▶ Built-In Strap Drive Circuits	▶ Set/Reset and Offset Strap Drivers for Degaussing, Self Test, and Offset Compensation
▶ I^{2}C Digital Interface	▶ Popular Two-Wire Serial Data Interface for Consumer Electronics
▶ Lead Free Package Construction	▶ RoHS Compliance
▶ Wide Magnetic Field Range (+/-8 Oe)	▶ Sensors Can Be Used in Strong Magnetic Field Environments with a 1° to 2° Degree Compass Heading Accuracy
▶ Software and Algorithm Support Available	▶ Compassing Heading, Hard Iron, Soft Iron, and Auto Calibration Libraries Available
▶ Fast 160 Hz Maximum Output Rate	▶ Enables Pedestrian Navigation and LBS Applications

SPECIFICATIONS (* Tested at 25°C except stated otherwise.)

Characteristics	Conditions*	Min	Typ	Max	Units
Power Supply					
Supply Voltage	VDD Referenced to AGND VDDIO Referenced to DGND	2.16 1.71	2.5 1.8	3.6 VDD+0.1	Volts Volts
Average Current Draw	Idle Mode Measurement Mode (7.5 Hz ODR; No measurement average, MA1:MA0 = 00) VDD = 2.5V, VDDIO = 1.8V (Dual Supply) VDD = VDDIO = 2.5V (Single Supply)	- -	2 100	- -	μA μA
Performance					
Field Range	Full scale (FS)	-8		+8	gauss
Mag Dynamic Range	3-bit gain control	±1		±8	gauss
Sensitivity (Gain)	VDD=3.0V, GN=0 to 7, 12-bit ADC	230		1370	LSb/gauss
Digital Resolution	VDD=3.0V, GN=0 to 7, 1-LSb, 12-bit ADC	0.73		4.35	milli-gauss
Noise Floor (Field Resolution)	VDD=3.0V, GN=0, No measurement average, Standard Deviation 100 samples (See typical performance graphs below)		2		milli-gauss
Linearity	±2.0 gauss input range			0.1	±% FS
Hysteresis	±2.0 gauss input range		±25		ppm
Cross-Axis Sensitivity	Test Conditions: Cross field = 0.5 gauss, Happlied = ±3 gauss		±0.2%		%FS/gauss
Output Rate (ODR)	Continuous Measurment Mode Single Measurement Mode	0.75		75 160	Hz Hz
Measurement Period	From receiving command to data ready		6		ms
Turn-on Time	Ready for I2C commands Analog Circuit Ready for Measurements		200 50		μs ms
Gain Tolerance	All gain/dynamic range settings		±5		%
I^2C Address	8-bit read address 8-bit write address		0x3D 0x3C		hex hex
I^2C Rate	Controlled by I^2C Master			400	kHz
I^2C Hysteresis	Hysteresis of Schmitt trigger inputs on SCL and SDA - Fall (VDDIO=1.8V) Rise (VDDIO=1.8V)		0.2*VDDIO 0.8*VDDIO		Volts Volts
Self Test	X & Y Axes Z Axis		±1.16 ±1.08		gauss
	X & Y & Z Axes (GN=5) Positive Bias X & Y & Z Axes (GN=5) Negative Bias	243 -575		575 -243	LSb
Sensitivity Tempco	T_A = -40 to 125°C, Uncompensated Output		-0.3		%/°C
General					
ESD Voltage	Human Body Model (all pins) Charged Device Model (all pins)			2000 750	Volts
Operating Temperature	Ambient	-30		85	°C
Storage Temperature	Ambient, unbiased	-40		125	°C

Characteristics	Conditions*	Min	Typ	Max	Units
Reflow Classification	MSL 3, 260 °C Peak Temperature				
Package Size	Length and Width	2.85	3.00	3.15	mm
Package Height		0.8	0.9	1.0	mm
Package Weight			18		mg

Absolute Maximum Ratings (* Tested at 25°C except stated otherwise.)

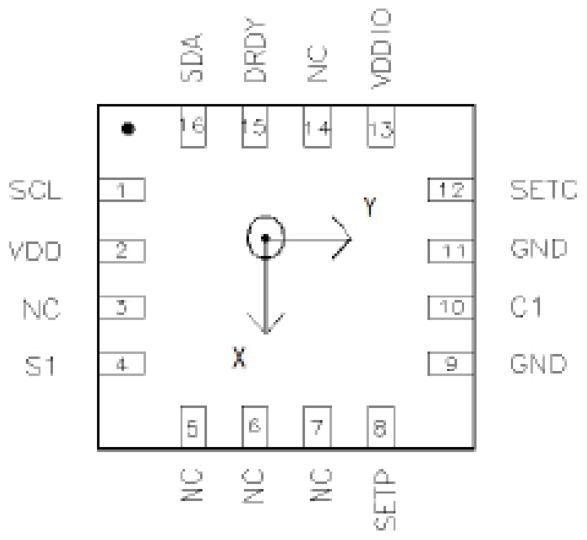
Characteristics	Min	Max	Units
Supply Voltage VDD	-0.3	4.8	Volts
Supply Voltage VDDIO	-0.3	4.8	Volts

PIN CONFIGURATIONS

Pin	Name	Description
1	SCL	Serial Clock – I^2C Master/Slave Clock
2	VDD	Power Supply (2.16V to 3.6V)
3	NC	Not to be Connected
4	S1	Tie to VDDIO
5	NC	Not to be Connected
6	NC	Not to be Connected
7	NC	Not to be Connected
8	SETP	Set/Reset Strap Positive – S/R Capacitor (C2) Connection
9	GND	Supply Ground
10	C1	Reservoir Capacitor (C1) Connection
11	GND	Supply Ground
12	SETC	S/R Capacitor (C2) Connection – Driver Side
13	VDDIO	IO Power Supply (1.71V to VDD)
14	NC	Not to be Connected
15	DRDY	Data Ready, Interrupt Pin. Internally pulled high. Optional connection. Low for 250 μsec when data is placed in the data output registers.
16	SDA	Serial Data – I^2C Master/Slave Data

Table 1: Pin Configurations

HMC5883L

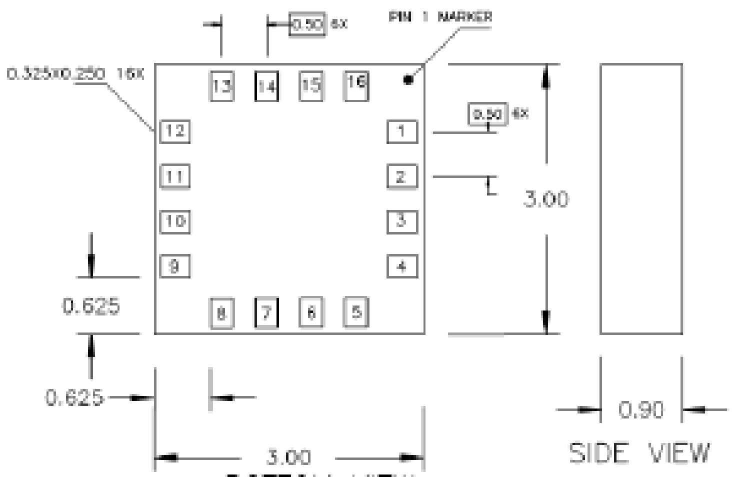


TOP VIEW (looking through)

Arrow indicates direction of magnetic field that generates a positive output reading in Normal Measurement configuration.

PACKAGE OUTLINES

PACKAGE DRAWING HMC5883L (16-PIN LPCC, dimensions in millimeters)

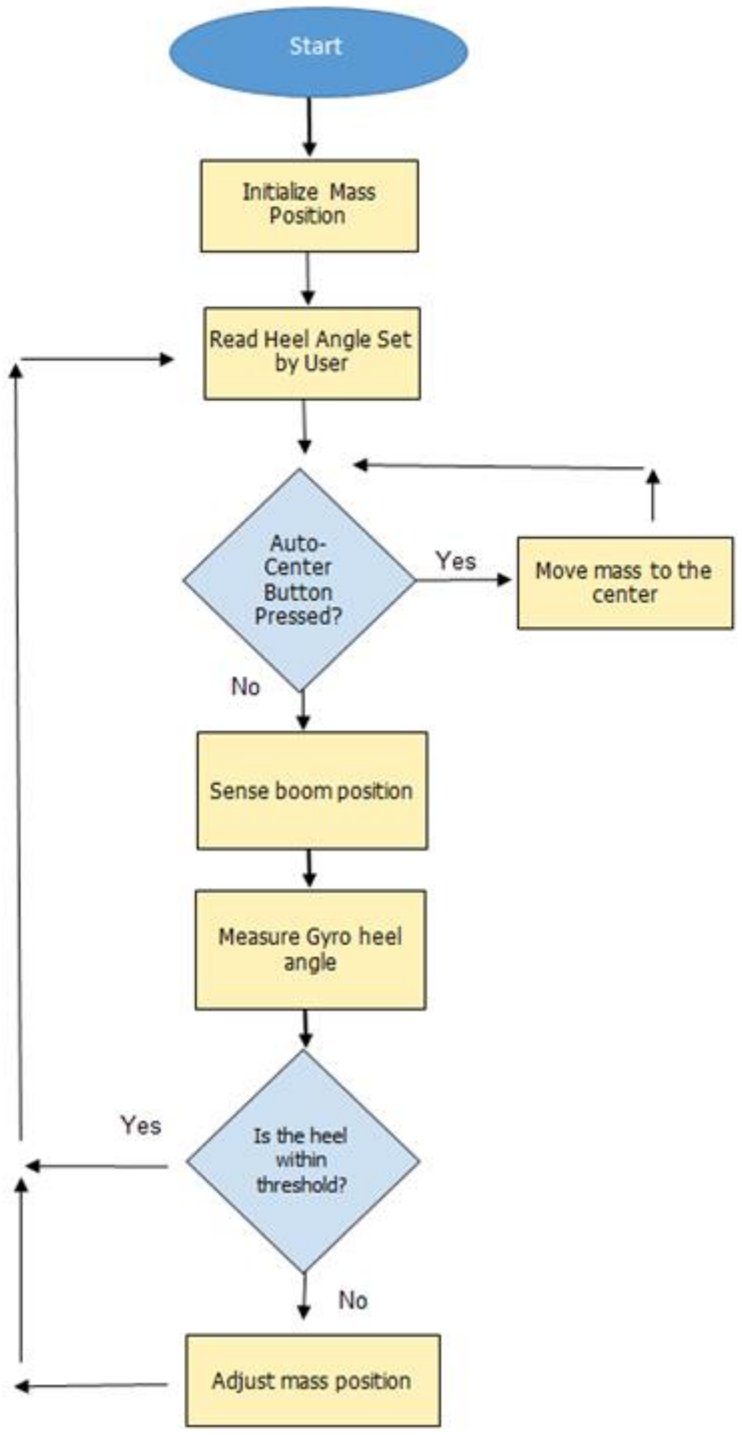


BOTTOM VIEW
(Dimensions in mm)

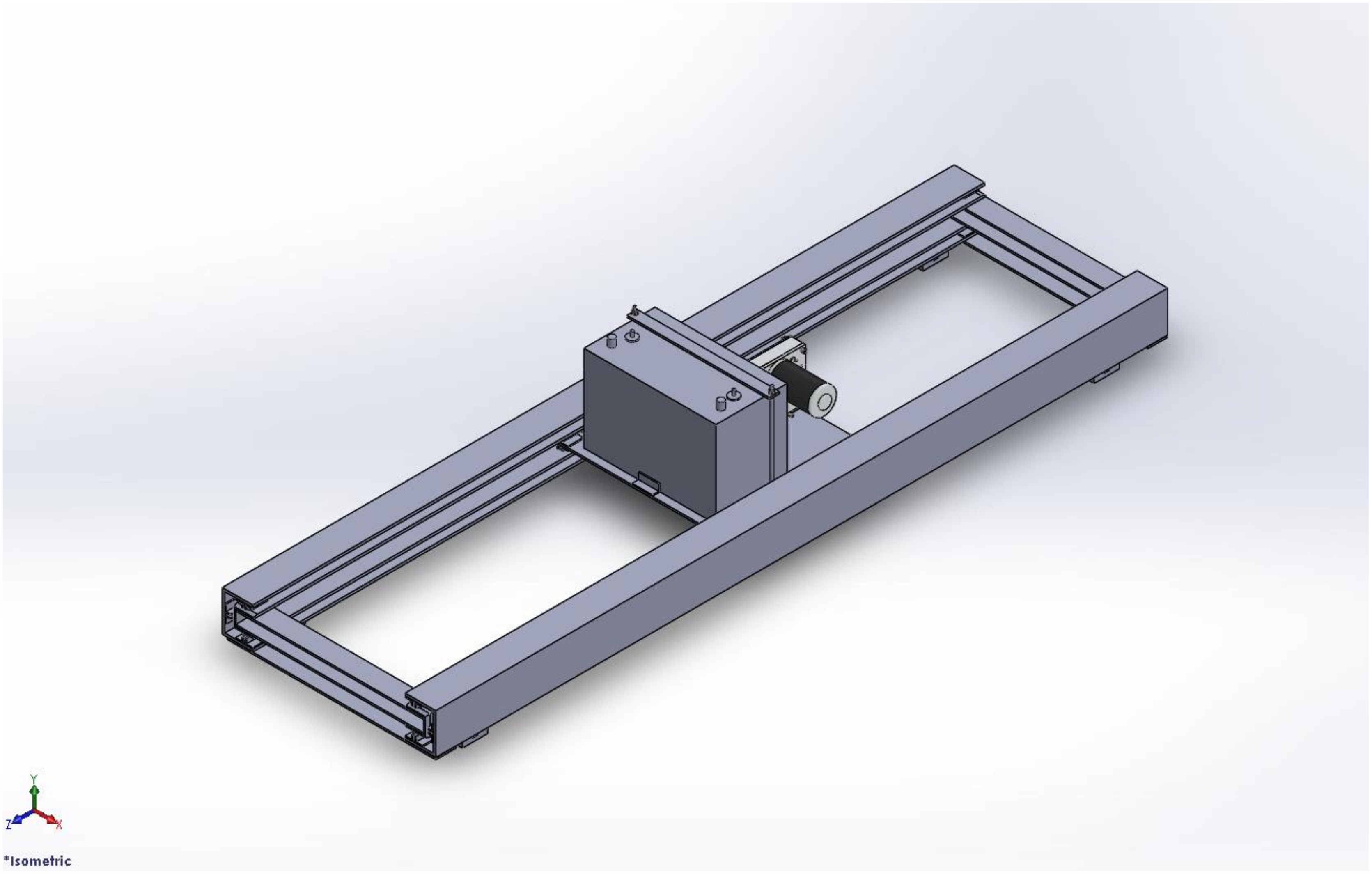
MOUNTING CONSIDERATIONS

The following is the recommend printed circuit board (PCB) footprint for the HMC5883L.

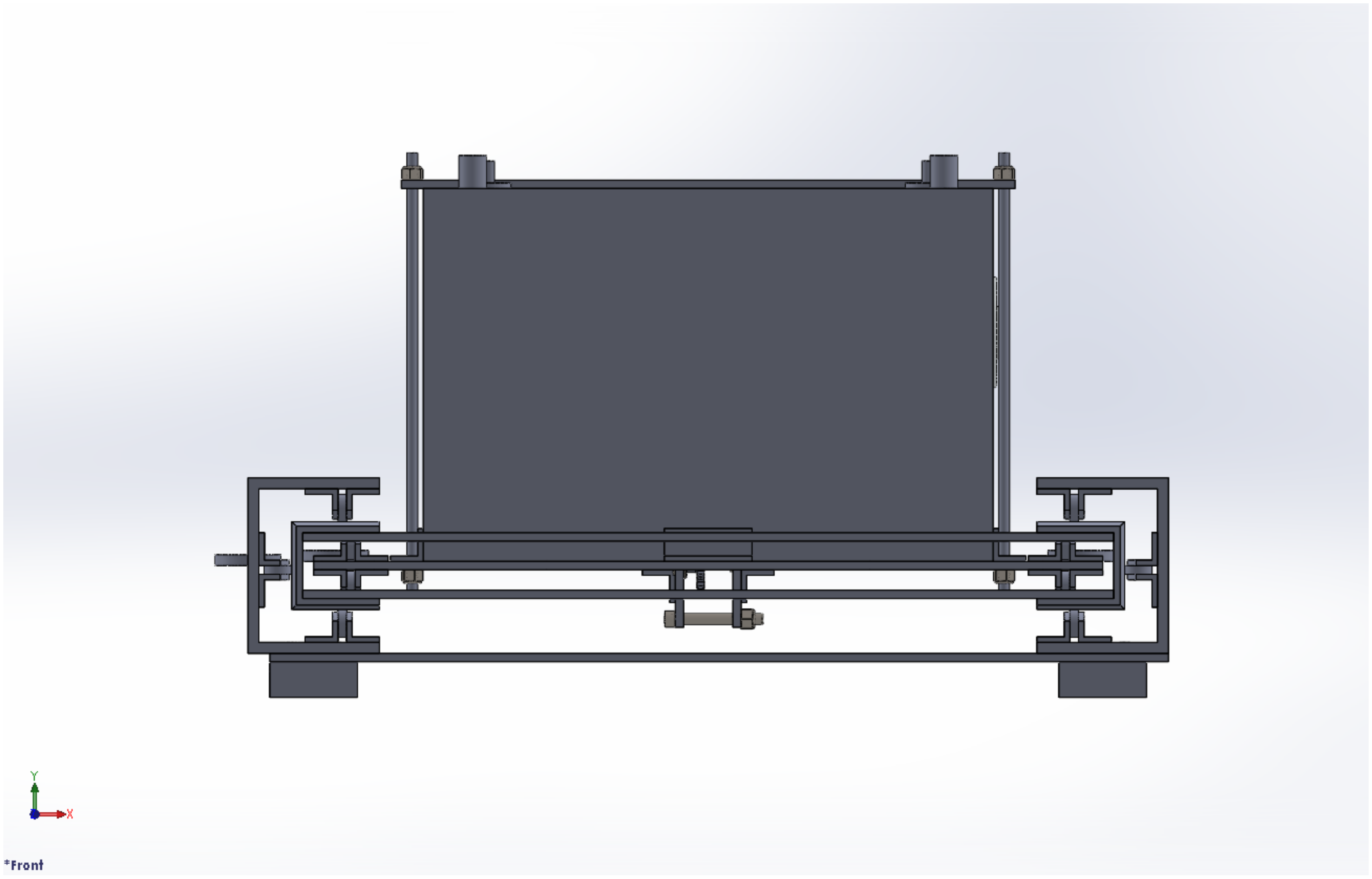
Appendix E


Start
Initialize Mass Position
Read Heel Angle Set by User
Auto-Center Button Pressed?
Yes
Move mass to the center
No
Sense boom position
Measure Gyro heel angle
Is the heel within threshold?
Yes
No
Adjust mass position

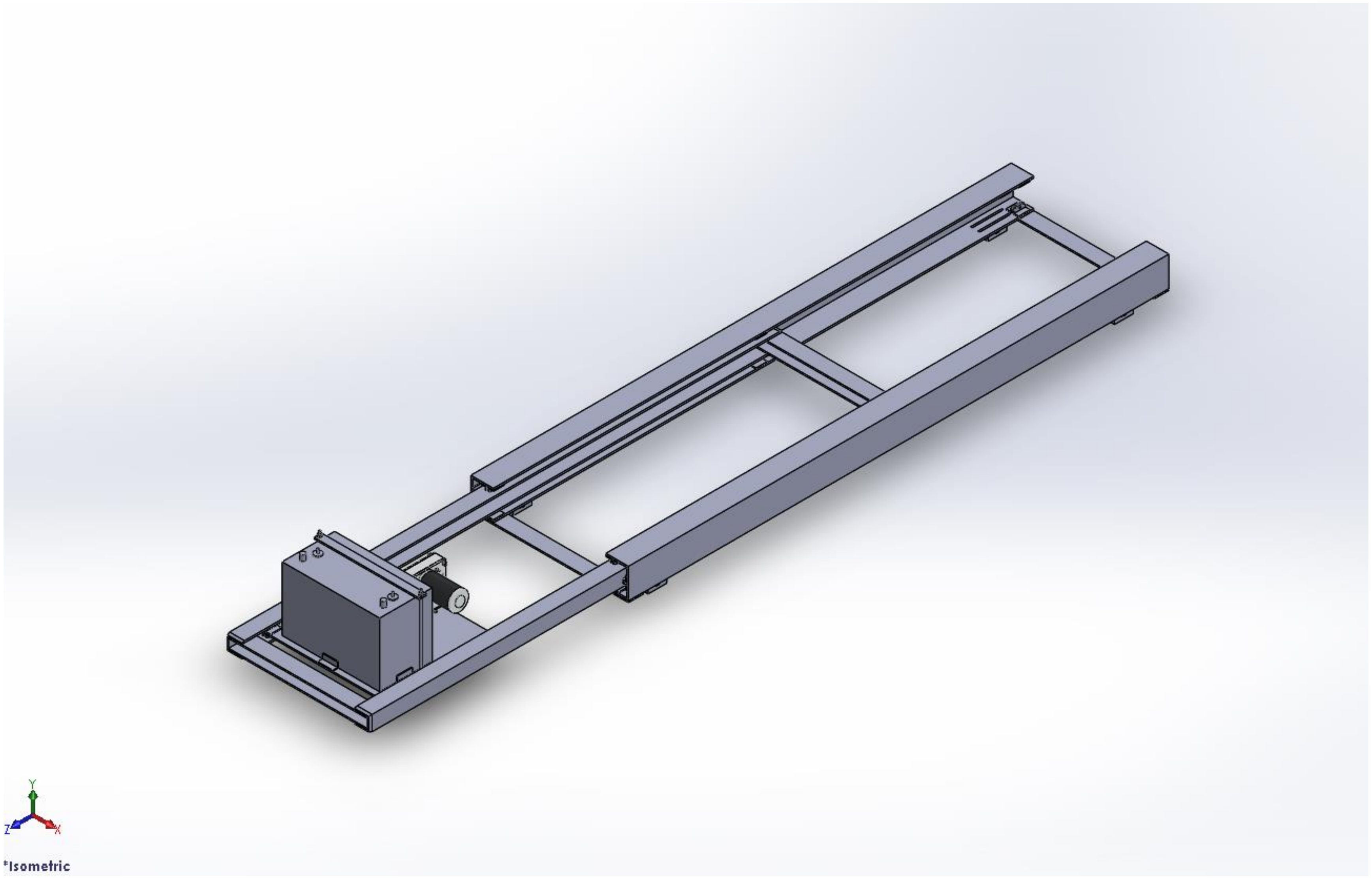
Appendix F



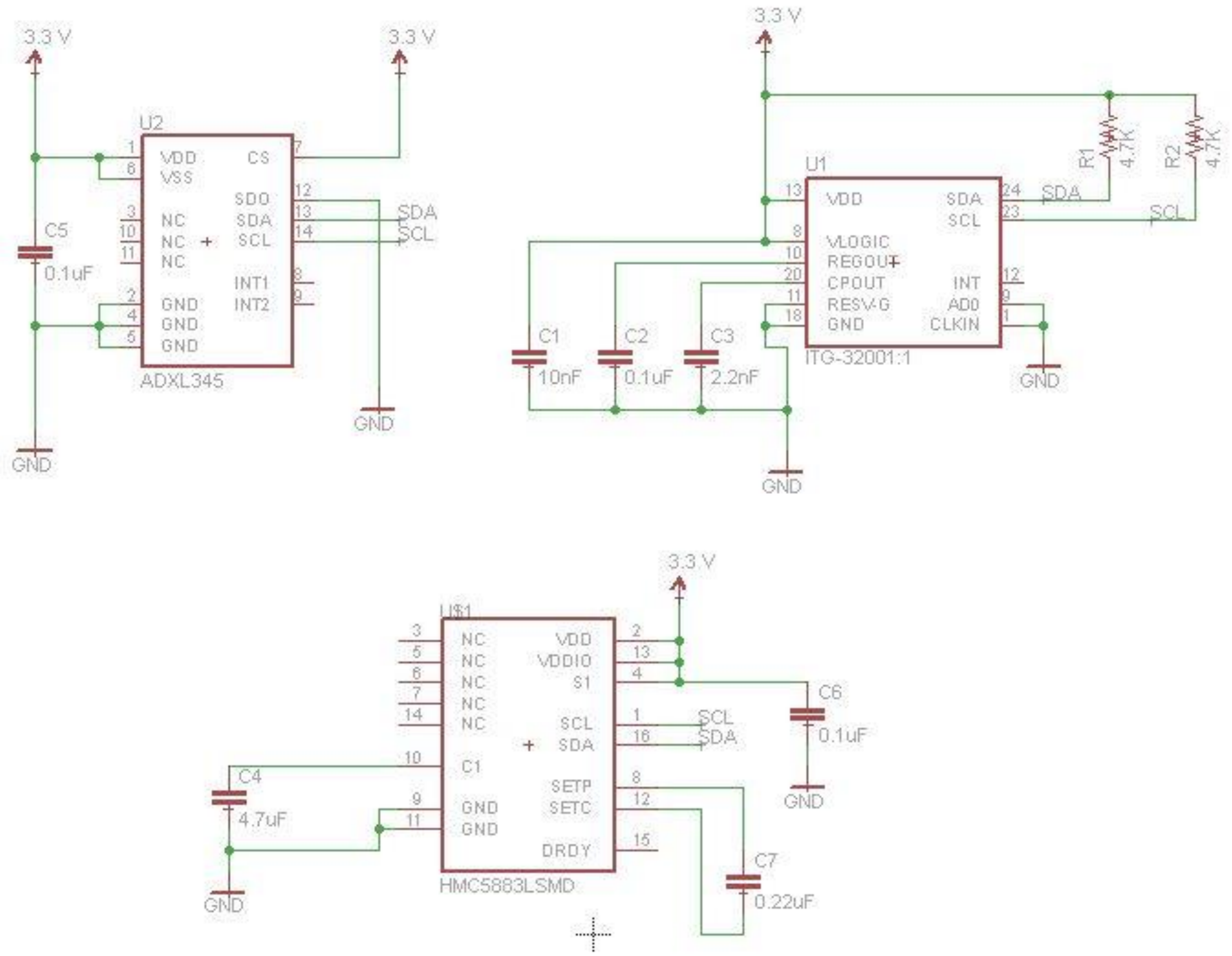
Appendix G

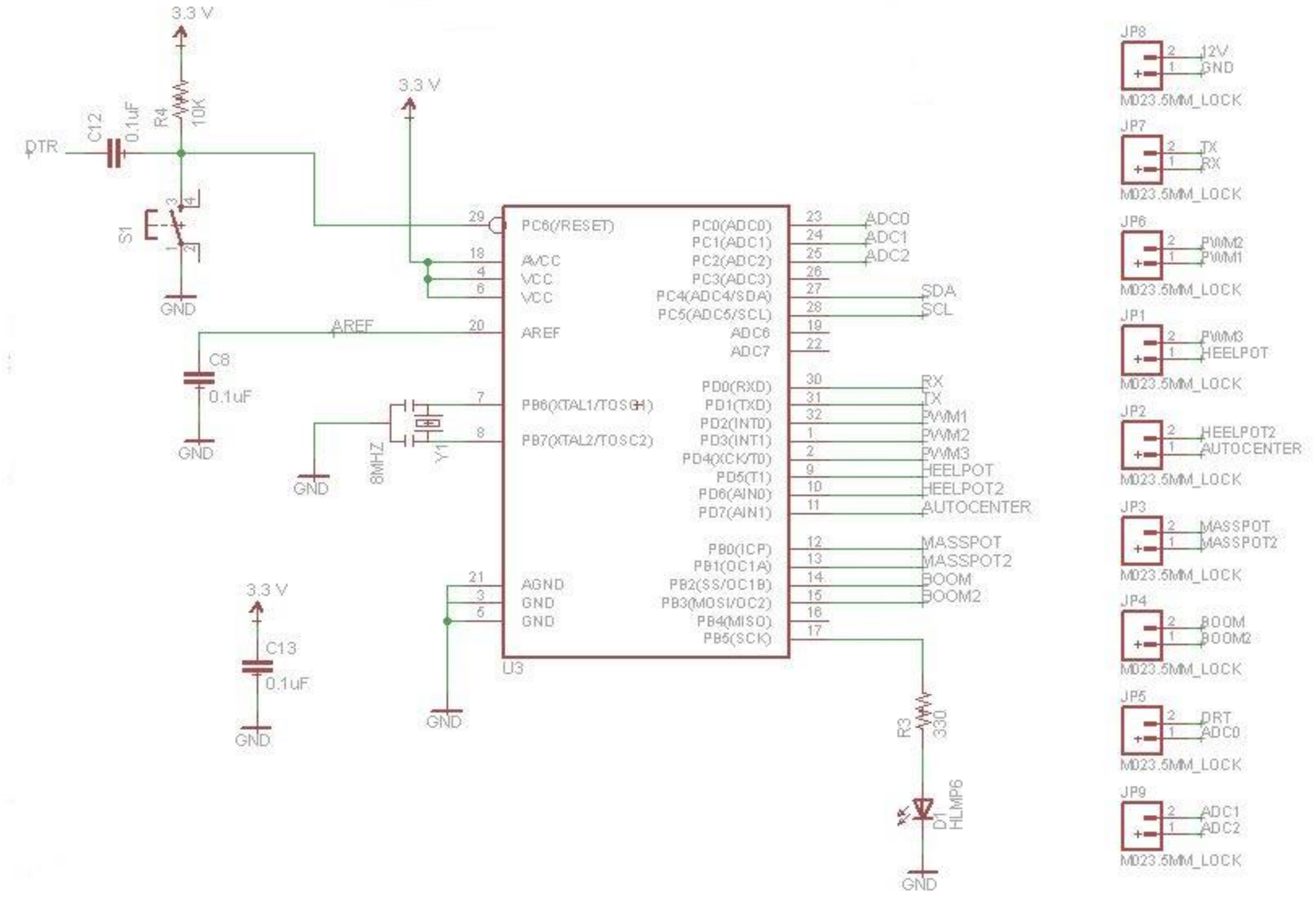


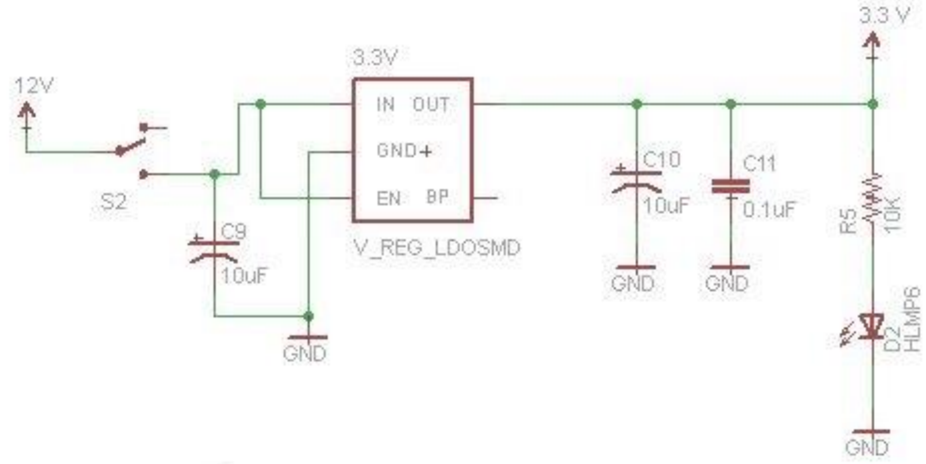
Appendix H



Appendix I


3.3 V
3.3 V
U2
VDD
VSS
CS
SDO
SDA
SCL
NC
NC
NC
INT1
INT2
GND
GND
GND
C5
0.1uF
ADXL345
SDA
SCL
GND
GND
3.3 V
U1
VDD
SDA
SCL
VLOGIC
REGOUT
CPOUT
RESV-G
GND
INT
AD0
CLKIN
ITG-32001:1
R1
4.7K
R2
4.7K
SDA
SCL
C1
10nF
C2
0.1uF
C3
2.2nF
GND
GND
3.3 V
U$1
NC
NC
NC
NC
NC
VDD
VDDIO
S1
SCL
SDA
C1
GND
GND
SETP
SETC
DRDY
HMC5883LSMD
SCL
SDA
C4
4.7uF
C6
0.1uF
C7
0.22uF
GND
GND

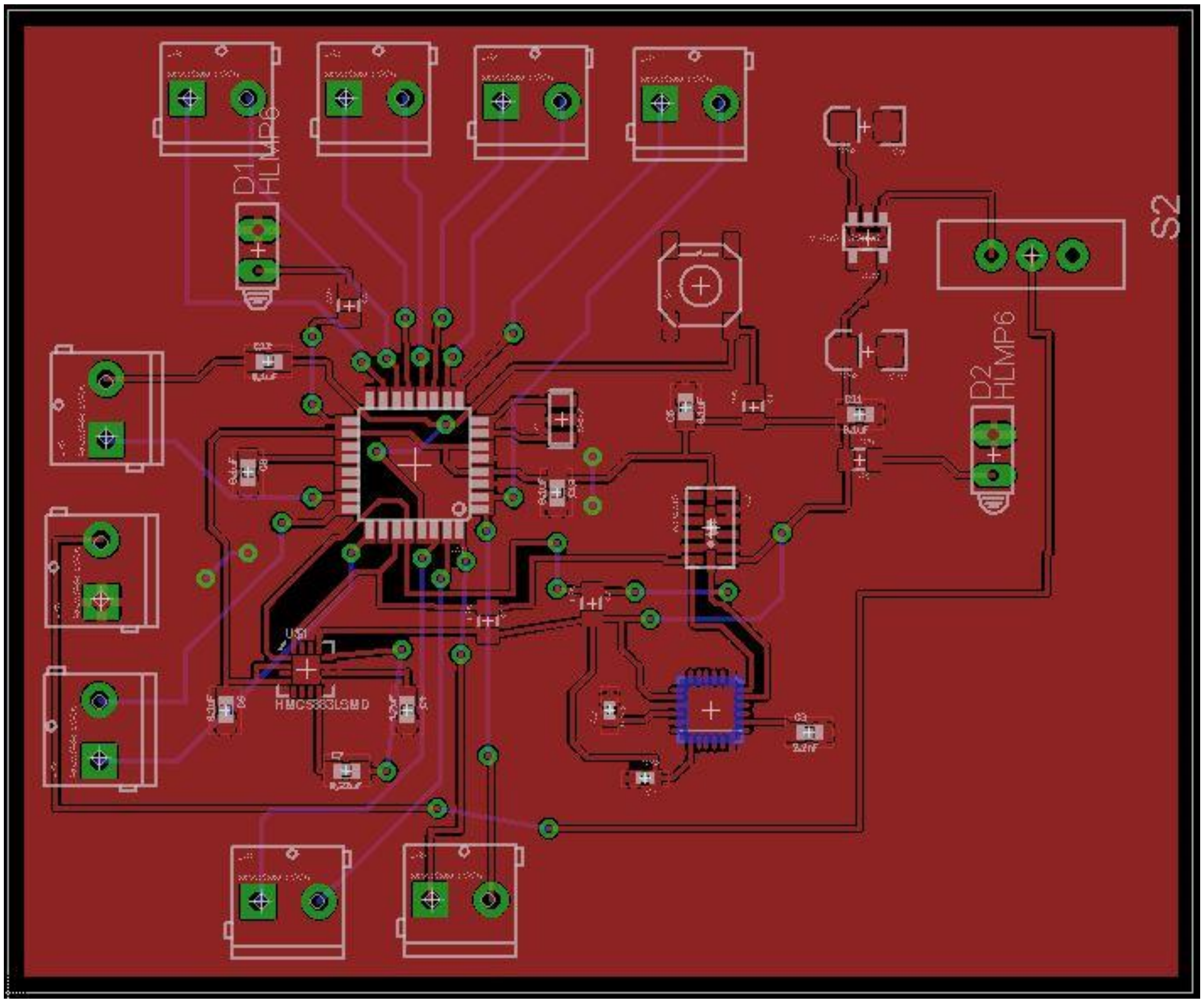

3.3 V
R4
10K
C12
0.1uF
DTR
S1
GND
3.3 V
PC6(/RESET)
AVCC
VCC
VCC
AREF
PB6(XTAL1/TOSC1)
PB7(XTAL2/TOSC2)
AGND
GND
GND
U3
PC0(ADC0)
PC1(ADC1)
PC2(ADC2)
PC3(ADC3)
PC4(ADC4/SDA)
PC5(ADC5/SCL)
ADC6
ADC7
PD0(RXD)
PD1(TXD)
PD2(INT0)
PD3(INT1)
PD4(XCK/T0)
PD5(T1)
PD6(AIN0)
PD7(AIN1)
PB0(ICP)
PB1(OC1A)
PB2(SS/OC1B)
PB3(MOSI/OC2)
PB4(MISO)
PB5(SCK)
ADC0
ADC1
ADC2
SDA
SCL
RX
TX
PWM1
PWM2
PWM3
HEELPOT
HEELPOT2
AUTOCENTER
MASSPOT
MASSPOT2
BOOM
BOOM2
AREF
C8
0.1uF
8MHZ
Y1
C13
0.1uF
R3
330
D1
HLMP6
JP8
12V
GND
JP7
TX
RX
JP6
PWM2
PWM1
JP1
PWM3
HEELPOT
JP2
HEELPOT2
AUTOCENTER
JP3
MASSPOT
MASSPOT2
JP4
BOOM
BOOM2
JP5
DRT
ADC0
JP9
ADC1
ADC2
M023.5MM_LOCK


3.3 V
3.3V
12V
IN OUT
GND
EN BP
S2
C9
10uF
V_REG_LDOSMD
C10
10uF
C11
0.1uF
R5
10K
GND
GND
GND
D2
HLMP6
GND

Appendix K

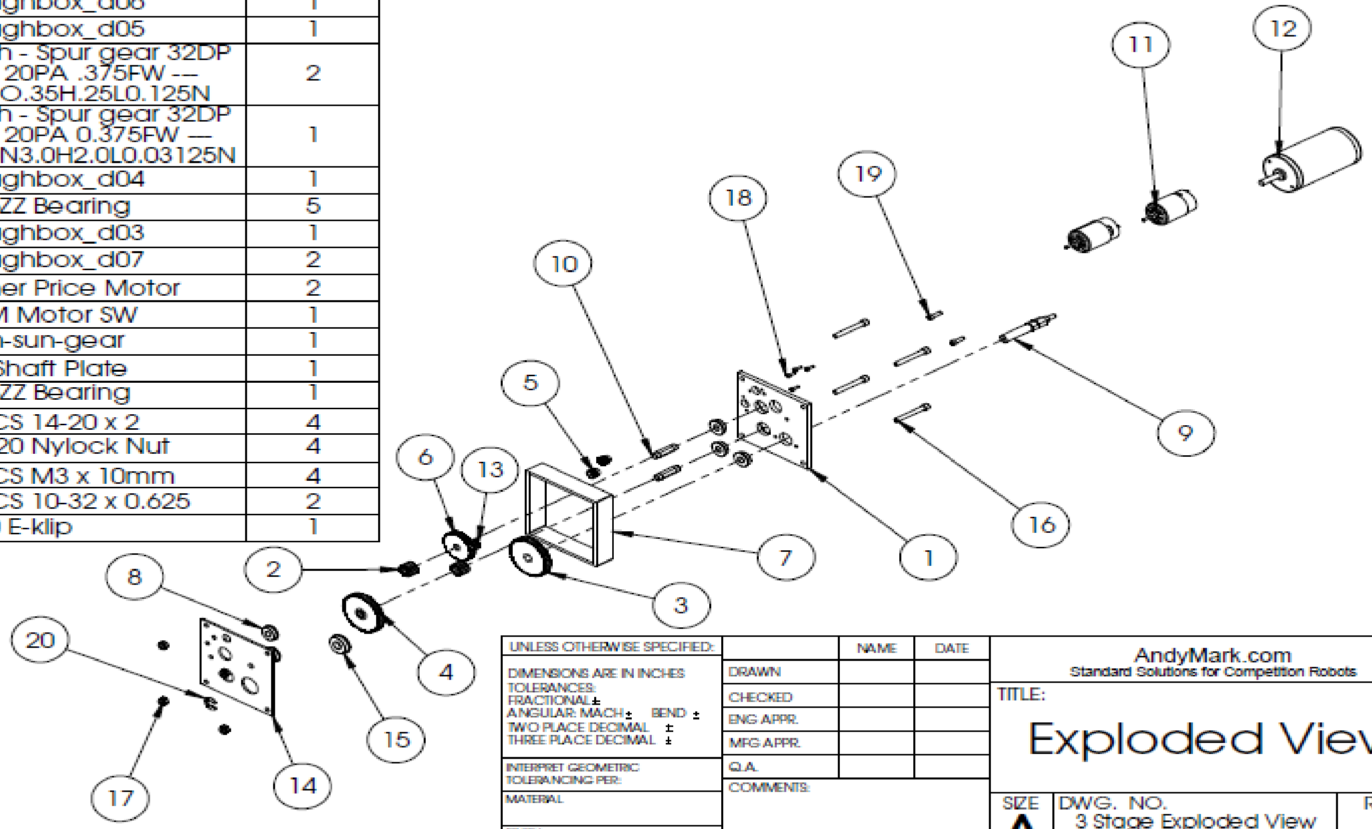
```
Partlist exported from C:/Users/Pedro/Desktop/Razor_final.sch

Part Value         Device                     Package                    Description

3.3V V_REG_LDOSMD  V_REG_LDOSMD               SOT23-5                    Voltage Regulator LDO
C1   10nF          10NF-CER                   0402-CAP                   10nF/.01uF/10000pF ceramic SMT
C2   0.1uF         .1UF-CER                   0402-CAP                   .1uF ceramic SMT
C3   2.2nF         2.2NF/2200PF-50V-10%(0603) 0603-CAP                   CAP-07877
C4   4.7uF         4.7UF-6.3V-10%(0603)0603   0603-CAP                   CAP-08280
C5   0.1uF         0.1UF-25V(□20%)(0603)      0603-CAP                   Ceramic
C6   0.1uF         0.1UF-25V(□20%)(0603)      0603-CAP                   Ceramic
C7   0.22uF        0.22UF-25V-10%(0603)       0603-CAP                   CAP-07822
C8   0.1uF         0.1UF-25V(□20%)(0603)      0603-CAP                   Ceramic
C9   10uF          CAP_POL1206                EIA3216                    Capacitor Polarized
C10  10uF          CAP_POL1206                EIA3216                    Capacitor Polarized
C11  0.1uF         0.1UF-25V(□20%)(0603)      0603-CAP                   Ceramic
C12  0.1uF         0.1UF-25V(□20%)(0603)      0603-CAP                   Ceramic
C13  0.1uF         0.1UF-25V(□20%)(0603)      0603-CAP                   Ceramic
D1   HLMP6         HLMP6                      HLMP6                      LED
D2   HLMP6         HLMP6                      HLMP6                      LED
JP1  M023.5MM_LOCK M023.5MM_LOCK              SCREWTERMINAL-3.5MM-2_LOCK Standard 2-pin 0.1" header. Use with
JP2  M023.5MM_LOCK M023.5MM_LOCK              SCREWTERMINAL-3.5MM-2_LOCK Standard 2-pin 0.1" header. Use with
JP3  M023.5MM_LOCK M023.5MM_LOCK              SCREWTERMINAL-3.5MM-2_LOCK Standard 2-pin 0.1" header. Use with
JP4  M023.5MM_LOCK M023.5MM_LOCK              SCREWTERMINAL-3.5MM-2_LOCK Standard 2-pin 0.1" header. Use with
JP5  M023.5MM_LOCK M023.5MM_LOCK              SCREWTERMINAL-3.5MM-2_LOCK Standard 2-pin 0.1" header. Use with
JP6  M023.5MM_LOCK M023.5MM_LOCK              SCREWTERMINAL-3.5MM-2_LOCK Standard 2-pin 0.1" header. Use with
JP7  M023.5MM_LOCK M023.5MM_LOCK              SCREWTERMINAL-3.5MM-2_LOCK Standard 2-pin 0.1" header. Use with
JP8  M023.5MM_LOCK M023.5MM_LOCK              SCREWTERMINAL-3.5MM-2_LOCK Standard 2-pin 0.1" header. Use with
JP9  M023.5MM_LOCK M023.5MM_LOCK              SCREWTERMINAL-3.5MM-2_LOCK Standard 2-pin 0.1" header. Use with
R1   4.7K          RESISTOR0805-RES           0805                       Resistor
R2   4.7K          RESISTOR0805-RES           0805                       Resistor
R3   330           RESISTOR0805-RES           0805                       Resistor
R4   10K           RESISTOR0805-RES           0805                       Resistor
R5   10K           RESISTOR0805-RES           0805                       Resistor
S1                 TAC_SWITCHSMD              TACTILE_SWITCH_SMD         Momentary Switch
S2                 SWITCH-SPDTPTH             SWITCH-SPDT                SPDT Switch
U$1  HMC5883LSMD   HMC5883LSMD                16LPCC                     3 Axis Digital Compass IC
U1   ITG-32001:1   ITG-32001:1                QFN-24_ITG3200_1:1         ITG-3200 3-axis gyro, digital output, qfn24
U2   ADXL345       ADXL345                    LGA14                      3-axis SPI/I2C 2/4/8/16g accelerometer
U3                 ATMEGA328_SMT              TQFP32-08                  32-Pin Atmega328 part
Y1   8MHZ          RESONATORSMHZ              RESONATOR-SMD              Resonator
```

Appendix J



Appendix L

ITEM NO.	PART NUMBER	Default/ QTY.
1	3S Motor Plate	1
2	Toughbox_d08	2
3	Toughbox_d06	1
4	Toughbox_d05	1
5	Inch - Spur gear 32DP 15T 20PA .375FW --- S15O.35H.25L0.125N	2
6	Inch - Spur gear 32DP 60T 20PA 0.375FW --- S60N3.0H2.0L0.03125N	1
7	Toughbox_d04	1
8	FR6ZZ Bearing	5
9	Toughbox_d03	1
10	Toughbox_d07	2
11	Fisher Price Motor	2
12	CIM Motor SW	1
13	cim-sun-gear	1
14	3S Shaft Plate	1
15	FR8ZZ Bearing	1
16	SHCS 14-20 x 2	4
17	14-20 Nylock Nut	4
18	SHCS M3 x 10mm	4
19	SHCS 10-32 x 0.625	2
20	500 E-klip	1



UNLESS OTHERWISE SPECIFIED:		NAME	DATE
DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
	CHECKED		
	ENG APPR.		
	MFG APPR.		
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
MATERIAL	COMMENTS:		
FINISH			
DO NOT SCALE DRAWING			

AndyMark.com
Standard Solutions for Competition Robots

TITLE:

Exploded View

SIZE	DWG. NO.	REV
A	3 Stage Exploded View	
SCALE: 1:24	WEIGHT:	SHEET 1 OF 1

5 4 3 2 1

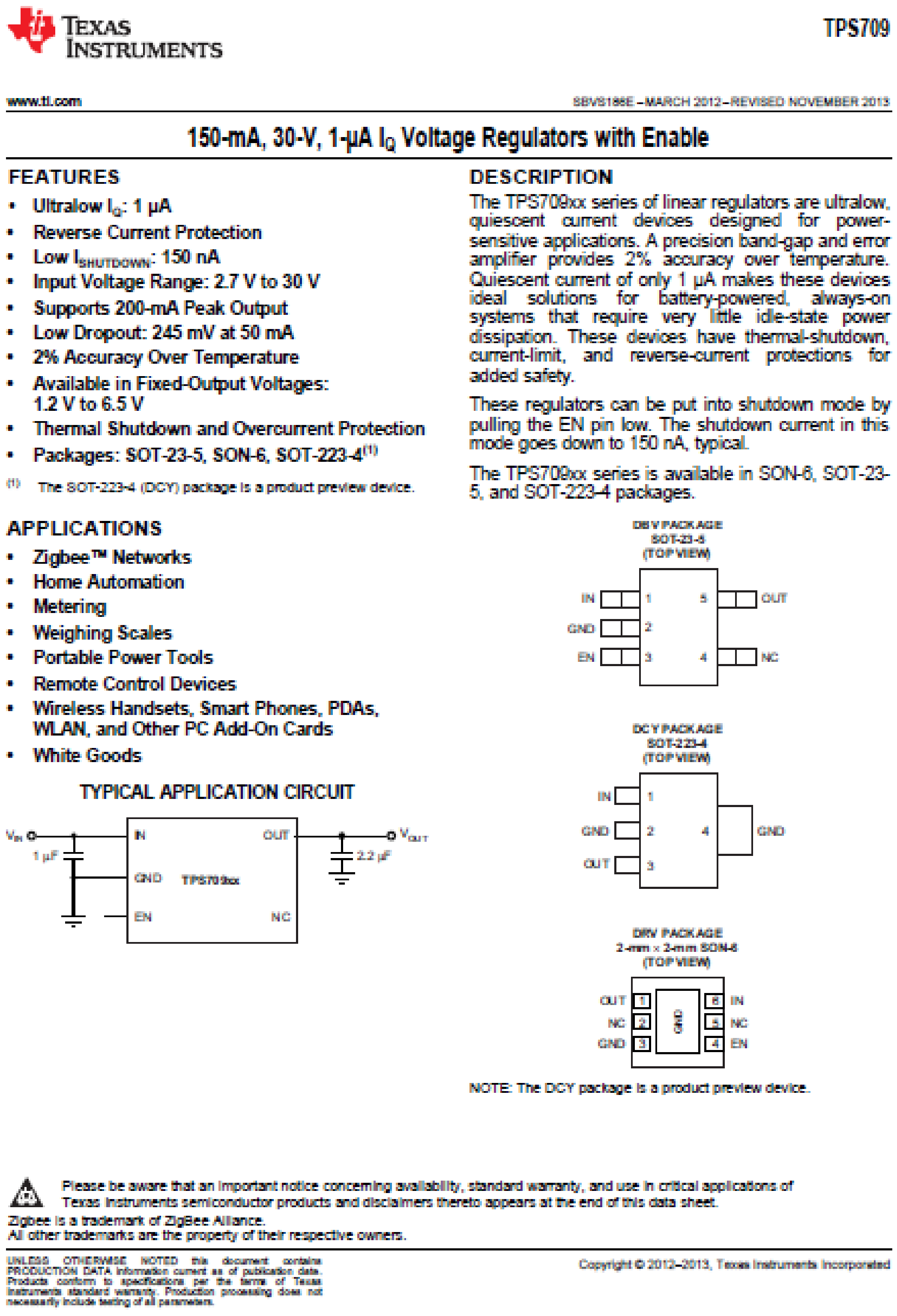
Appendix M

```
/*****************************************************************************************
******
* Test Sketch for Razor AHRS v1.4.2
* 9 Degree of Measurement Attitude and Heading Reference System
* for Sparkfun "9DOF Razor IMU" and "9DOF Sensor Stick"
*
* Released under GNU GPL (General Public License) v3.0
* Copyright (C) 2013 Peter Bartz [http://ptrbrtz.net]
* Copyright (C) 2011-2012 Quality & Usability Lab, Deutsche Telekom Laboratories, TU Berlin
* Written by Peter Bartz (peter-bartz@gmx.de)
*
* Infos, updates, bug reports, contributions and feedback:
* https://github.com/ptrbrtz/razor-9dof-ahrs
*****************************************************************************************
*****/

/*
NOTE: There seems to be a bug with the serial library in Processing versions 1.5
and 1.5.1: "WARNING: RXTX Version mismatch ...".
Processing 2.0.x seems to work just fine. Later versions may too.
Alternatively, the older version 1.2.1 also works and is still available on the web.
*/

import processing.opengl.*;
import processing.serial.*;

// IF THE SKETCH CRASHES OR HANGS ON STARTUP, MAKE SURE YOU ARE USING THE RIGHT SERIAL
PORT:
// 1. Have a look at the Processing console output of this sketch.
// 2. Look for the serial port list and find the port you need (it's the same as in Arduino).
// 3. Set your port number here:
final static int SERIAL_PORT_NUM = 0;
// 4. Try again.


final static int SERIAL_PORT_BAUD_RATE = 57700;

float yaw = 0.0f;
float pitch = 0.0f;
float roll = 0.0f;
float yawOffset = 200.0f;

PFont font;
Serial serial;
```

```
boolean synched = false;

void drawArrow(float headWidthFactor, float headLengthFactor) {
  float headWidth = headWidthFactor * 200.0f;
  float headLength = headLengthFactor * 200.0f;

  pushMatrix();

  // Draw base
  translate(0, 0, -100);
  box(100, 100, 200);

  // Draw pointer
  translate(-headWidth/2, -50, -100);
  beginShape(QUAD_STRIP);
    vertex(0, 0 ,0);
    vertex(0, 100, 0);
    vertex(headWidth, 0 ,0);
    vertex(headWidth, 100, 0);
    vertex(headWidth/2, 0, -headLength);
    vertex(headWidth/2, 100, -headLength);
    vertex(0, 0 ,0);
    vertex(0, 100, 0);
  endShape();
  beginShape(TRIANGLES);
    vertex(0, 0, 0);
    vertex(headWidth, 0, 0);
    vertex(headWidth/2, 0, -headLength);
    vertex(0, 100, 0);
    vertex(headWidth, 100, 0);
    vertex(headWidth/2, 100, -headLength);
  endShape();

  popMatrix();
}

void drawBoard() {
  pushMatrix();

  rotateY(-radians(yaw - yawOffset));
  rotateX(-radians(pitch));
  rotateZ(radians(roll));

  // Board body
  fill(255, 0, 0);
  box(250, 20, 400);

  // Forward-arrow
  pushMatrix();
  translate(0, 0, -200);
  scale(0.5f, 0.2f, 0.25f);
  fill(0, 255, 0);
  drawArrow(1.0f, 2.0f);
  popMatrix();

  popMatrix();
}

// Skip incoming serial stream data until token is found
boolean readToken(Serial serial, String token) {
  // Wait until enough bytes are available
  if (serial.available() < token.length())
    return false;

  // Check if incoming bytes match token
  for (int i = 0; i < token.length(); i++) {
    if (serial.read() != token.charAt(i))
      return false;
  }

  return true;
}

// Global setup
void setup() {
  // Setup graphics
  size(640, 480, OPENGL);
  smooth();
  noStroke();
  frameRate(50);

  // Load font
  font = loadFont("Univers-66.vlw");
  textFont(font);

  // Setup serial port I/O
  println("AVAILABLE SERIAL PORTS:");
  println(Serial.list());
  String portName = Serial.list()[SERIAL_PORT_NUM];
  println();
  println("HAVE A LOOK AT THE LIST ABOVE AND SET THE RIGHT SERIAL PORT NUMBER IN THE CODE!");
  println(" -> Using port " + SERIAL_PORT_NUM + ": " + portName);
  serial = new Serial(this, portName, SERIAL_PORT_BAUD_RATE);
}

void setupRazor() {
  println("Trying to setup and synch Razor...");

  // On Mac OSX and Linux (Windows too?) the board will do a reset when we connect, which is really
bad.
  // See "Automatic (Software) Reset" on http://www.arduino.cc/en/Main/ArduinoBoardProMini
  // So we have to wait until the bootloader is finished and the Razor firmware can receive commands.
  // To prevent this, disconnect/cut/unplug the DTR line going to the board. This also has the advantage,
  // that the angles you receive are stable right from the beginning.
  delay(3000); // 3 seconds should be enough

  // Set Razor output parameters
  serial.write("#ob"); // Turn on binary output
  serial.write("#o1"); // Turn on continuous streaming output
  serial.write("#oe0"); // Disable error message output

  // Synch with Razor
  serial.clear(); // Clear input buffer up to here
  serial.write("#s00"); // Request synch token
}

float readFloat(Serial s) {
  // Convert from little endian (Razor) to big endian (Java) and interpret as float
  return Float.intBitsToFloat(s.read() + (s.read() << 8) + (s.read() << 16) + (s.read() << 24));
}

void draw() {
   // Reset scene
  background(0);
  lights();

  // Sync with Razor
  if (!synched) {
    textAlign(CENTER);
    fill(255);
    text("Connecting to Razor...", width/2, height/2, -200);

    if (frameCount == 2)
      setupRazor(); // Set ouput params and request synch token
    else if (frameCount > 2)
      synched = readToken(serial, "#SYNCH00\r\n"); // Look for synch token
    return;
  }

  // Read angles from serial port
  while (serial.available() >= 12) {
    yaw = readFloat(serial);
    pitch = readFloat(serial);
    roll = readFloat(serial);
```

```
 }

 // Draw board
 pushMatrix();
 translate(width/2, height/2, -350);
 drawBoard();
 popMatrix();

 textFont(font, 20);
 fill(255);
 textAlign(LEFT);

 // Output info text
 text("Point FTDI connector towards screen and press 'a' to align", 10, 25);

 // Output angles
 pushMatrix();
 translate(10, height - 10);
 textAlign(LEFT);
 text("Yaw: " + ((int) yaw), 0, 0);
 text("Pitch: " + ((int) pitch), 150, 0);
 text("Roll: " + ((int) roll), 300, 0);
 popMatrix();
}

void keyPressed() {
 switch (key) {
  case '0': // Turn Razor's continuous output stream off
   serial.write("#o0");
   break;
  case '1': // Turn Razor's continuous output stream on
   serial.write("#o1");
   break;
  case 'f': // Request one single yaw/pitch/roll frame from Razor (use when continuous streaming is off)
   serial.write("#f");
   break;
  case 'a': // Align screen with Razor
   yawOffset = yaw;
 }
}
```

Appendix N



TPS709

www.ti.com SBVS186E – MARCH 2012 – REVISED NOVEMBER 2013

150-mA, 30-V, 1-µA I_Q Voltage Regulators with Enable

FEATURES

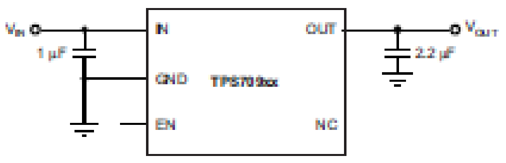
- Ultralow I_Q: 1 µA
- Reverse Current Protection
- Low $I_{SHUTDOWN}$: 150 nA
- Input Voltage Range: 2.7 V to 30 V
- Supports 200-mA Peak Output
- Low Dropout: 245 mV at 50 mA
- 2% Accuracy Over Temperature
- Available in Fixed-Output Voltages: 1.2 V to 6.5 V
- Thermal Shutdown and Overcurrent Protection
- Packages: SOT-23-5, SON-6, SOT-223-4[1]

[1] The SOT-223-4 (DCY) package is a product preview device.

APPLICATIONS

- Zigbee™ Networks
- Home Automation
- Metering
- Weighing Scales
- Portable Power Tools
- Remote Control Devices
- Wireless Handsets, Smart Phones, PDAs, WLAN, and Other PC Add-On Cards
- White Goods

TYPICAL APPLICATION CIRCUIT

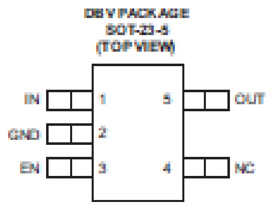


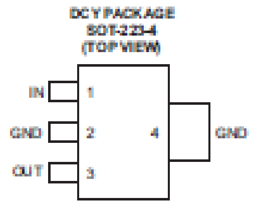
DESCRIPTION

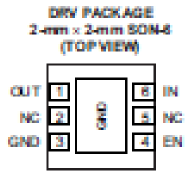
The TPS709xx series of linear regulators are ultralow, quiescent current devices designed for power-sensitive applications. A precision band-gap and error amplifier provides 2% accuracy over temperature. Quiescent current of only 1 µA makes these devices ideal solutions for battery-powered, always-on systems that require very little idle-state power dissipation. These devices have thermal-shutdown, current-limit, and reverse-current protections for added safety.

These regulators can be put into shutdown mode by pulling the EN pin low. The shutdown current in this mode goes down to 150 nA, typical.

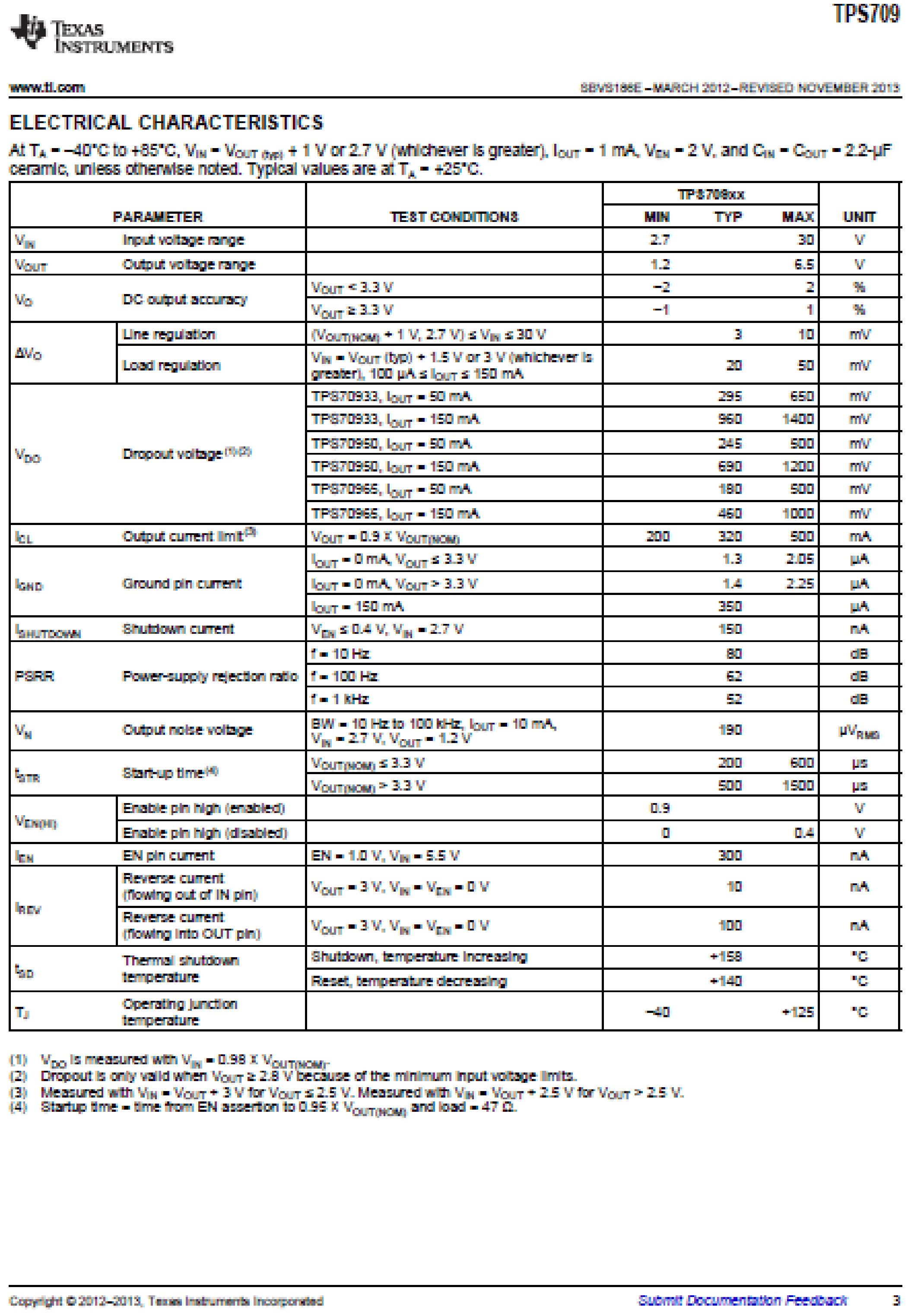
The TPS709xx series is available in SON-6, SOT-23-5, and SOT-223-4 packages.







NOTE: The DCY package is a product preview device.



TEXAS INSTRUMENTS

TPS709

www.ti.com

SBVS186E –MARCH 2012–REVISED NOVEMBER 2013

ELECTRICAL CHARACTERISTICS

At T_A = –40°C to +85°C, V_{IN} = $V_{OUT\,(typ)}$ + 1 V or 2.7 V (whichever is greater), I_{OUT} = 1 mA, V_{EN} = 2 V, and C_{IN} = C_{OUT} = 2.2-µF ceramic, unless otherwise noted. Typical values are at T_A = +25°C.

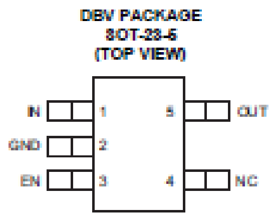
PARAMETER		TEST CONDITIONS	TPS709xx			UNIT
			MIN	TYP	MAX	
V_{IN}	Input voltage range		2.7		30	V
V_{OUT}	Output voltage range		1.2		6.5	V
V_O	DC output accuracy	V_{OUT} < 3.3 V	–2		2	%
		V_{OUT} ≥ 3.3 V	–1		1	%
ΔV_O	Line regulation	($V_{OUT(NOM)}$ + 1 V, 2.7 V) ≤ V_{IN} ≤ 30 V		3	10	mV
	Load regulation	V_{IN} = V_{OUT} (typ) + 1.5 V or 3 V (whichever is greater), 100 µA ≤ I_{OUT} ≤ 150 mA		20	50	mV
V_{DO}	Dropout voltage (1)(2)	TPS70933, I_{OUT} = 50 mA		295	650	mV
		TPS70933, I_{OUT} = 150 mA		960	1400	mV
		TPS70950, I_{OUT} = 50 mA		245	500	mV
		TPS70950, I_{OUT} = 150 mA		690	1200	mV
		TPS70965, I_{OUT} = 50 mA		180	500	mV
		TPS70965, I_{OUT} = 150 mA		460	1000	mV
I_{CL}	Output current limit (3)	V_{OUT} = 0.9 X $V_{OUT(NOM)}$	200	320	500	mA
I_{GND}	Ground pin current	I_{OUT} = 0 mA, V_{OUT} ≤ 3.3 V		1.3	2.05	µA
		I_{OUT} = 0 mA, V_{OUT} > 3.3 V		1.4	2.25	µA
		I_{OUT} = 150 mA		350		µA
$I_{SHUTDOWN}$	Shutdown current	V_{EN} ≤ 0.4 V, V_{IN} = 2.7 V		150		nA
PSRR	Power-supply rejection ratio	f = 10 Hz		80		dB
		f = 100 Hz		62		dB
		f = 1 kHz		52		dB
V_N	Output noise voltage	BW = 10 Hz to 100 kHz, I_{OUT} = 10 mA, V_{IN} = 2.7 V, V_{OUT} = 1.2 V		190		$µV_{RMS}$
t_{STR}	Start-up time (4)	$V_{OUT(NOM)}$ ≤ 3.3 V		200	600	µs
		$V_{OUT(NOM)}$ > 3.3 V		500	1500	µs
$V_{EN(HI)}$	Enable pin high (enabled)		0.9			V
	Enable pin high (disabled)		0		0.4	V
I_{EN}	EN pin current	EN = 1.0 V, V_{IN} = 5.5 V		300		nA
I_{REV}	Reverse current (flowing out of IN pin)	V_{OUT} = 3 V, V_{IN} = V_{EN} = 0 V		10		nA
	Reverse current (flowing into OUT pin)	V_{OUT} = 3 V, V_{IN} = V_{EN} = 0 V		100		nA
t_{SD}	Thermal shutdown temperature	Shutdown, temperature increasing		+158		°C
		Reset, temperature decreasing		+140		°C
T_J	Operating junction temperature		–40		+125	°C

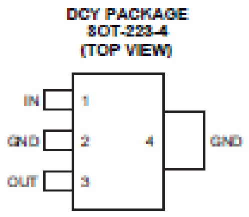
(1) V_{DO} is measured with V_{IN} = 0.98 X $V_{OUT(NOM)}$.
(2) Dropout is only valid when V_{OUT} ≥ 2.8 V because of the minimum input voltage limits.
(3) Measured with V_{IN} = V_{OUT} + 3 V for V_{OUT} ≤ 2.5 V. Measured with V_{IN} = V_{OUT} + 2.5 V for V_{OUT} > 2.5 V.
(4) Startup time = time from EN assertion to 0.95 X $V_{OUT(NOM)}$ and load = 47 Ω.

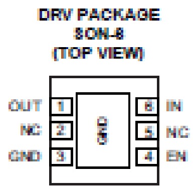
Copyright © 2012–2013, Texas Instruments Incorporated

Submit Documentation Feedback 3

PIN CONFIGURATIONS







NOTE: The DCY package is a product preview device.

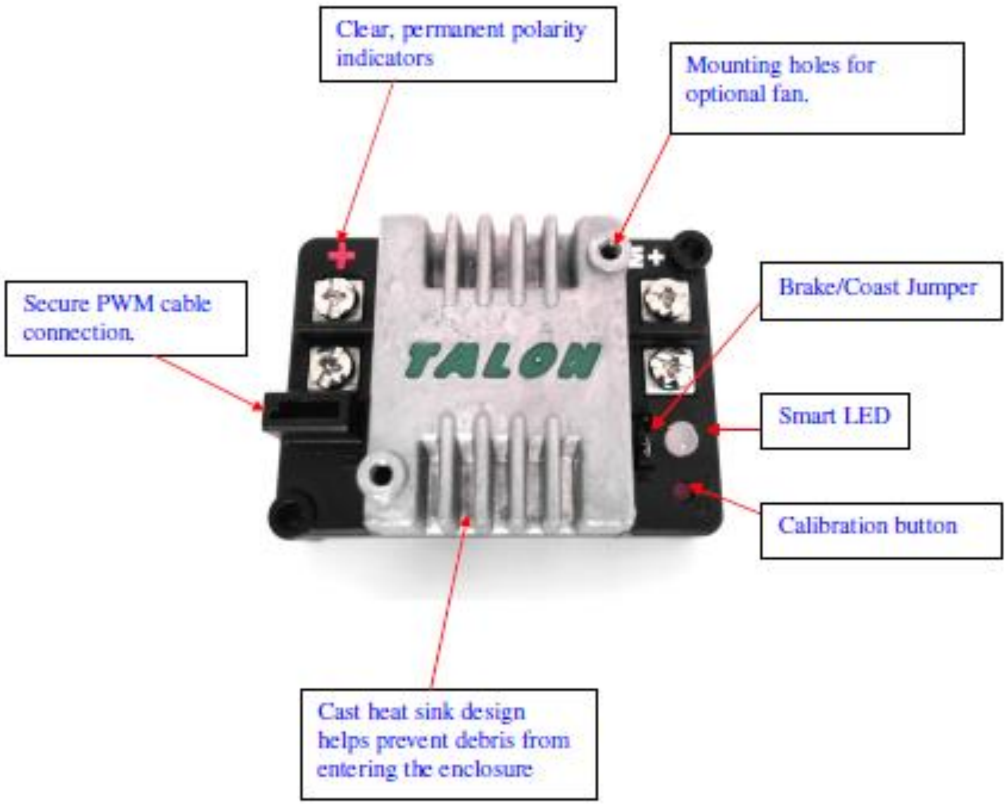
PIN DESCRIPTIONS

PIN NAME	PIN NO.			DESCRIPTION
	DBV	DCY	DRV	
EN	3	—	4	Enable pin. Driving this pin high enables the device. Driving this pin low puts the device into low current shutdown. This pin has an internal pull-up resistor and can be left floating to enable the device.
GND	2	2, 4	3	Ground
IN	1	1	6	Unregulated input to the device
NC	4	—	2, 5	No internal connection
OUT	5	3	1	Regulated output voltage. A small 2.2-µF or greater ceramic capacitor should be connected from this pin to ground to assure stability.

Appendix O

Talon 4/2/2013

Device Overview


Clear, permanent polarity indicators
Mounting holes for optional fan.
Secure PWM cable connection.
Brake/Coast Jumper
Smart LED
Calibration button
Cast heat sink design helps prevent debris from entering the enclosure
TALON

Cross The Road Electronics, LLC Page 2 4/2/2013

1) What is a Talon?

Both the Talon and Talon SR are devices used to control the rotational velocity (speed) of a brushed DC motor through modulating power over time. The differences between the Talon and Talon SR are indicated in Orange text throughout the manual.

2) Features

- Passive cooling design (heatsink)
- Conformal coating
- Locked-antiphase rectification (Talon)
- Synchronous sign magnitude rectification (Talon SR)
- Lightweight small foot print
- Smart LED, blinks proportional to throttle.
- 15 khz switching frequency
- Metal chip resistant
- 6-28 volt DC input
- Up to 100 amps peak 60 amps continuous current.
- Mounting holes to allow for optional 40 mm fan.
- Secure PWM connection
- Simple calibration
- 10-bit input and output precision
- User selectable brake/coast
- 4% neutral dead band
- Linear throttle response

Appendix P

```
/**************************************************************************************************************
* Razor AHRS Firmware v1.4.2
* 9 Degree of Measurement Attitude and Heading Reference System
* for Sparkfun "9DOF Razor IMU" (SEN-10125 and SEN-10736)
* and "9DOF Sensor Stick" (SEN-10183, 10321 and SEN-10724)
*
* Released under GNU GPL (General Public License) v3.0
* Copyright (C) 2013 Peter Bartz [http://ptrbrtz.net]
* Copyright (C) 2011-2012 Quality & Usability Lab, Deutsche Telekom Laboratories, TU Berlin
*
* Infos, updates, bug reports, contributions and feedback:
*     https://github.com/ptrbrtz/razor-9dof-ahrs
*
*
* History:
*   * Original code (http://code.google.com/p/sf9domahrs/) by Doug Weibel and Jose Julio,
*     based on ArduIMU v1.5 by Jordi Munoz and William Premerlani, Jose Julio and Doug Weibel. Thank you!
*
*   * Updated code (http://groups.google.com/group/sf_9dof_ahrs_update) by David Malik (david.zsolt.malik@gmail.com)
*     for new Sparkfun 9DOF Razor hardware (SEN-10125).
*
*   * Updated and extended by Peter Bartz (peter-bartz@gmx.de):
*     * v1.3.0
*       * Cleaned up, streamlined and restructured most of the code to make it more comprehensible.
*       * Added sensor calibration (improves precision and responsiveness a lot!).
*       * Added binary yaw/pitch/roll output.
*       * Added basic serial command interface to set output modes/calibrate sensors/synch stream/etc.
*       * Added support to synch automatically when using Rovering Networks Bluetooth modules (and compatible).
*       * Wrote new easier to use test program (using Processing).
*       * Added support for new version of "9DOF Razor IMU": SEN-10736.
*       --> The output of this code is not compatible with the older versions!
*       --> A Processing sketch to test the tracker is available.
*     * v1.3.1
*       * Initializing rotation matrix based on start-up sensor readings -> orientation OK right away.
*       * Adjusted gyro low-pass filter and output rate settings.
*     * v1.3.2
*       * Adapted code to work with new Arduino 1.0 (and older versions still).
*     * v1.3.3
*       * Improved synching.
*     * v1.4.0
*       * Added support for SparkFun "9DOF Sensor Stick" (versions SEN-10183, SEN-10321 and SEN-10724).
*     * v1.4.1
*       * Added output modes to read raw and/or calibrated sensor data in text or binary format.
*       * Added static magnetometer soft iron distortion compensation
*     * v1.4.2
*       * (No core firmware changes)
*
* TODOs:
*   * Allow optional use of EEPROM for storing and reading calibration values.
*   * Use self-test and temperature-compensation features of the sensors.
**************************************************************************************************************/
#include <avr/io.h>
#include <PID_v1.h>
#include <math.h>

#include <Servo.h>

Servo myservo;

/*
  "9DOF Razor IMU" hardware versions: SEN-10125 and SEN-10736
  ATMega328@3.3V, 8MHz

  ADXL345  : Accelerometer
  HMC5843  : Magnetometer on SEN-10125
  HMC5883L : Magnetometer on SEN-10736
  ITG-3200 : Gyro

  Arduino IDE : Select board "Arduino Pro or Pro Mini (3.3v, 8Mhz) w/ATmega328"
*/


/*
  Axis definition (differs from definition printed on the board!):
    X axis pointing forward (towards the short edge with the connector holes)
    Y axis pointing to the right
    and Z axis pointing down.

  Positive yaw   : clockwise
  Positive roll  : right wing down
  Positive pitch : nose up

  Transformation order: first yaw then pitch then roll.
*/

/*
  Serial commands that the firmware understands:

  "#o<params>" - Set OUTPUT mode and parameters. The available options are:

    // Streaming output
    "#o0" - DISABLE continuous streaming output. Also see #f below.
    "#o1" - ENABLE continuous streaming output.

    // Angles output
    "#ob" - Output angles in BINARY format (yaw/pitch/roll as binary float, so one output frame
        is 3x4 = 12 bytes long).
    "#ot" - Output angles in TEXT format (Output frames have form like "#YPR=-142.28,-5.38,33.52",
        followed by carriage return and line feed [\r\n]).

    // Sensor calibration
    "#oc" - Go to CALIBRATION output mode.
    "#on" - When in calibration mode, go on to calibrate NEXT sensor.

    // Sensor data output
    "#osct" - Output CALIBRATED SENSOR data of all 9 axes in TEXT format.
          One frame consist of three lines - one for each sensor: acc, mag, gyr.
    "#osrt" - Output RAW SENSOR data of all 9 axes in TEXT format.
          One frame consist of three lines - one for each sensor: acc, mag, gyr.
    "#osbt" - Output BOTH raw and calibrated SENSOR data of all 9 axes in TEXT format.
          One frame consist of six lines - like #osrt and #osct combined (first RAW, then CALIBRATED).
          NOTE: This is a lot of number-to-text conversion work for the little 8MHz chip on the Razor boards.
          In fact it's too much and an output frame rate of 50Hz can not be maintained. #osbb.
    "#oscb" - Output CALIBRATED SENSOR data of all 9 axes in BINARY format.
          One frame consist of three 3x3 float values = 36 bytes. Order is: acc x/y/z, mag x/y/z, gyr x/y/z.
    "#osrb" - Output RAW SENSOR data of all 9 axes in BINARY format.
          One frame consist of three 3x3 float values = 36 bytes. Order is: acc x/y/z, mag x/y/z, gyr x/y/z.
    "#osbb" - Output BOTH raw and calibrated SENSOR data of all 9 axes in BINARY format.
          One frame consist of 2x36 = 72 bytes - like #osrb and #oscb combined (first RAW, then CALIBRATED).

    // Error message output
    "#oe0" - Disable ERROR message output.
    "#oe1" - Enable ERROR message output.


  "#f" - Request one output frame - useful when continuous output is disabled and updates are
      required in larger intervals only. Though #f only requests one reply, replies are still
      bound to the internal 20ms (50Hz) time raster. So worst case delay that #f can add is 19.99ms.


  "#s<xy>" - Request synch token - useful to find out where the frame boundaries are in a continuous
      binary stream or to see if tracker is present and answering. The tracker will send
      "#SYNCH<xy>\r\n" in response (so it's possible to read using a readLine() function).
      x and y are two mandatory but arbitrary bytes that can be used to find out which request
      the answer belongs to.


  ("#C" and "#D" - Reserved for communication with optional Bluetooth module.)

  Newline characters are not required. So you could send "#ob#o1#s", which
  would set binary output mode, enable continuous streaming output and request
  a synch token all at once.

  The status LED will be on if streaming output is enabled and off otherwise.

  Byte order of binary output is little-endian: least significant byte comes first.
*/



/*****************************************************************/
/*********** USER SETUP AREA! Set your options here! *************/
/*****************************************************************/

// HARDWARE OPTIONS
/*****************************************************************/
// Select your hardware here by uncommenting one line!
//#define HW__VERSION_CODE 10125 // SparkFun "9DOF Razor IMU" version "SEN-10125" (HMC5843 magnetometer)
#define HW__VERSION_CODE 10736 // SparkFun "9DOF Razor IMU" version "SEN-10736" (HMC5883L magnetometer)
//#define HW__VERSION_CODE 10183 // SparkFun "9DOF Sensor Stick" version "SEN-10183" (HMC5843 magnetometer)
//#define HW__VERSION_CODE 10321 // SparkFun "9DOF Sensor Stick" version "SEN-10321" (HMC5843 magnetometer)
//#define HW__VERSION_CODE 10724 // SparkFun "9DOF Sensor Stick" version "SEN-10724" (HMC5883L magnetometer)


// OUTPUT OPTIONS
/*****************************************************************/
// Set your serial port baud rate used to send out data here!
#define OUTPUT__BAUD_RATE 57600

// Sensor data output interval in milliseconds
// This may not work, if faster than 20ms (=50Hz)
// Code is tuned for 20ms, so better leave it like that
#define OUTPUT__DATA_INTERVAL 20  // in milliseconds

// Output mode definitions (do not change)
#define OUTPUT__MODE_CALIBRATE_SENSORS 0 // Outputs sensor min/max values as text for manual calibration
#define OUTPUT__MODE_ANGLES 1 // Outputs yaw/pitch/roll in degrees
#define OUTPUT__MODE_SENSORS_CALIB 2 // Outputs calibrated sensor values for all 9 axes
#define OUTPUT__MODE_SENSORS_RAW 3 // Outputs raw (uncalibrated) sensor values for all 9 axes
#define OUTPUT__MODE_SENSORS_BOTH 4 // Outputs calibrated AND raw sensor values for all 9 axes
// Output format definitions (do not change)
#define OUTPUT__FORMAT_TEXT 0 // Outputs data as text
#define OUTPUT__FORMAT_BINARY 1 // Outputs data as binary float

// Select your startup output mode and format here!
int output_mode = OUTPUT__MODE_ANGLES;
int output_format = OUTPUT__FORMAT_TEXT;

// Select if serial continuous streaming output is enabled per default on startup.
#define OUTPUT__STARTUP_STREAM_ON true  // true or false

// If set true, an error message will be output if we fail to read sensor data.
// Message format: "!ERR: reading <sensor>", followed by "\r\n".
```

```
boolean output_errors = false;  // true or false

// Bluetooth
// You can set this to true, if you have a Rovering Networks Bluetooth Module attached.
// The connect/disconnect message prefix of the module has to be set to "#".
// (Refer to manual, it can be set like this: SO,#)
// When using this, streaming output will only be enabled as long as we're connected. That way
// receiver and sender are synchronzed easily just by connecting/disconnecting.
// It is not necessary to set this! It just makes life easier when writing code for
// the receiving side. The Processing test sketch also works without setting this.
// NOTE: When using this, OUTPUT__STARTUP_STREAM_ON has no effect!
#define OUTPUT__HAS_RN_BLUETOOTH false  // true or false


// SENSOR CALIBRATION
/*****************************************************************/
// How to calibrate? Read the tutorial at http://dev.qu.tu-berlin.de/projects/sf-razor-9dof-ahrs
// Put MIN/MAX and OFFSET readings for your board here!
// Accelerometer
// "accel x,y,z (min/max) = X_MIN/X_MAX  Y_MIN/Y_MAX  Z_MIN/Z_MAX"
#define ACCEL_X_MIN ((float) -281)
#define ACCEL_X_MAX ((float) 272)
#define ACCEL_Y_MIN ((float) -280)
#define ACCEL_Y_MAX ((float) 281)
#define ACCEL_Z_MIN ((float) -280)
#define ACCEL_Z_MAX ((float) 240)

// Magnetometer (standard calibration mode)
// "magn x,y,z (min/max) = X_MIN/X_MAX  Y_MIN/Y_MAX  Z_MIN/Z_MAX"
#define MAGN_X_MIN ((float) -460)
#define MAGN_X_MAX ((float) 669)
#define MAGN_Y_MIN ((float) -595)
#define MAGN_Y_MAX ((float) 545)
#define MAGN_Z_MIN ((float) -540)
#define MAGN_Z_MAX ((float)  766)

// Magnetometer (extended calibration mode)
// Uncommend to use extended magnetometer calibration (compensates hard & soft iron errors)
#define CALIBRATION__MAGN_USE_EXTENDED true
const float magn_ellipsoid_center[3] = {108.674, -42.9559, -38.7677};
const float magn_ellipsoid_transform[3][3] = {{0.948064, -.00572107, -0.0170356}, {-0.00572107, 0.989783, 0.0132464}, {-0.0170356,
0.0132464, 0.970555}};

// Gyroscope
// "gyro x,y,z (current/average) = .../OFFSET_X  .../OFFSET_Y  .../OFFSET_Z
#define GYRO_AVERAGE_OFFSET_X ((float) -5.07)
#define GYRO_AVERAGE_OFFSET_Y ((float) 22.40)
#define GYRO_AVERAGE_OFFSET_Z ((float) -10.05)

/*
// Calibration example:

// "accel x,y,z (min/max) = -277.00/264.00  -256.00/278.00  -299.00/235.00"
#define ACCEL_X_MIN ((float) -277)
#define ACCEL_X_MAX ((float) 264)
#define ACCEL_Y_MIN ((float) -256)
#define ACCEL_Y_MAX ((float) 278)
#define ACCEL_Z_MIN ((float) -299)
#define ACCEL_Z_MAX ((float) 235)

// "magn x,y,z (min/max) = -511.00/581.00  -516.00/568.00  -489.00/486.00"
//#define MAGN_X_MIN ((float) -511)
//#define MAGN_X_MAX ((float) 581)
//#define MAGN_Y_MIN ((float) -516)
//#define MAGN_Y_MAX ((float) 568)
//#define MAGN_Z_MIN ((float) -489)
```

```
//#define MAGN_Z_MAX ((float) 486)

// Extended magn
#define CALIBRATION__MAGN_USE_EXTENDED true
const float magn_ellipsoid_center[3] = {91.5, -13.5, -48.1};
const float magn_ellipsoid_transform[3][3] = {{0.902, -0.00354, 0.000636}, {-0.00354, 0.9, -0.00599}, {0.000636, -0.00599, 1}};

// Extended magn (with Sennheiser HD 485 headphones)
//#define CALIBRATION__MAGN_USE_EXTENDED true
//const float magn_ellipsoid_center[3] = {72.3360, 23.0954, 53.6261};
//const float magn_ellipsoid_transform[3][3] = {{0.879685, 0.000540833, -0.0106054}, {0.000540833, 0.891086, -0.0130338}, {-0.0106054, -
0.0130338, 0.997494}};

//"gyro x,y,z (current/average) = -40.00/-42.05  98.00/96.20  -18.00/-18.36"
#define GYRO_AVERAGE_OFFSET_X ((float) -42.05)
#define GYRO_AVERAGE_OFFSET_Y ((float) 96.20)
#define GYRO_AVERAGE_OFFSET_Z ((float) -18.36)
*/


// DEBUG OPTIONS
/*****************************************************************/
// When set to true, gyro drift correction will not be applied
#define DEBUG__NO_DRIFT_CORRECTION false
// Print elapsed time after each I/O loop
#define DEBUG__PRINT_LOOP_TIME false


/*****************************************************************/
/****************** END OF USER SETUP AREA!  *********************/
/*****************************************************************/







// Check if hardware version code is defined
#ifndef HW__VERSION_CODE
  // Generate compile error
  #error YOU HAVE TO SELECT THE HARDWARE YOU ARE USING! See "HARDWARE OPTIONS" in "USER SETUP AREA" at top of
Razor_AHRS.ino!
#endif

#include <Wire.h>

// Sensor calibration scale and offset values
#define ACCEL_X_OFFSET ((ACCEL_X_MIN + ACCEL_X_MAX) / 2.0f)
#define ACCEL_Y_OFFSET ((ACCEL_Y_MIN + ACCEL_Y_MAX) / 2.0f)
#define ACCEL_Z_OFFSET ((ACCEL_Z_MIN + ACCEL_Z_MAX) / 2.0f)
#define ACCEL_X_SCALE (GRAVITY / (ACCEL_X_MAX - ACCEL_X_OFFSET))
#define ACCEL_Y_SCALE (GRAVITY / (ACCEL_Y_MAX - ACCEL_Y_OFFSET))
#define ACCEL_Z_SCALE (GRAVITY / (ACCEL_Z_MAX - ACCEL_Z_OFFSET))

#define MAGN_X_OFFSET ((MAGN_X_MIN + MAGN_X_MAX) / 2.0f)
#define MAGN_Y_OFFSET ((MAGN_Y_MIN + MAGN_Y_MAX) / 2.0f)
#define MAGN_Z_OFFSET ((MAGN_Z_MIN + MAGN_Z_MAX) / 2.0f)
#define MAGN_X_SCALE (100.0f / (MAGN_X_MAX - MAGN_X_OFFSET))
#define MAGN_Y_SCALE (100.0f / (MAGN_Y_MAX - MAGN_Y_OFFSET))
#define MAGN_Z_SCALE (100.0f / (MAGN_Z_MAX - MAGN_Z_OFFSET))
```

```
// Gain for gyroscope (ITG-3200)
#define GYRO_GAIN 0.06957 // Same gain on all axes
#define GYRO_SCALED_RAD(x) (x * TO_RAD(GYRO_GAIN)) // Calculate the scaled gyro readings in radians per second

// DCM parameters
#define Kp_ROLLPITCH 0.05f
#define Ki_ROLLPITCH 0.0005f
#define Kp_YAW 1.0f
#define Ki_YAW 0.000002f

// Stuff
#define STATUS_LED_PIN 13  // Pin number of status LED
#define GRAVITY 256.0f // "1G reference" used for DCM filter and accelerometer calibration
#define TO_RAD(x) (x * 0.01745329252)  // *pi/180
#define TO_DEG(x) (x * 57.2957795131)  // *180/pi

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////
///////          MQP        ///////////////////////////////////////////////////////////////////////////////

//#define HEELSET_PIN 10   //Skipper desired heel angle
#define BOOM_PIN 14     //boom sensor which negates the HEELSET value for the motor
#define MASSPOS_PIN A1 // potentiometer wiper (middle terminal) connected to analog pin 24
                 // outside leads to ground and +5V
#define AUTOC_PIN 7
#define HEELSET_PIN A2


//Define Variables we'll be connecting to
//double Setpoint, Input, Output;
//Specify the links and initial tuning parameters
//float PID myPID(&Input, &Output, &Setpoint,2,5,1, INVERSE);

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////



// Sensor variables
float accel[3];  // Actually stores the NEGATED acceleration (equals gravity, if board not moving).
float accel_min[3];
float accel_max[3];

float magnetom[3];
float magnetom_min[3];
float magnetom_max[3];
float magnetom_tmp[3];

float gyro[3];
float gyro_average[3];
int gyro_num_samples = 0;

// DCM variables
float MAG_Heading;
float Accel_Vector[3]= {0, 0, 0}; // Store the acceleration in a vector
float Gyro_Vector[3]= {0, 0, 0}; // Store the gyros turn rate in a vector
float Omega_Vector[3]= {0, 0, 0}; // Corrected Gyro_Vector data
float Omega_P[3]= {0, 0, 0}; // Omega Proportional correction
float Omega_I[3]= {0, 0, 0}; // Omega Integrator
float Omega[3]= {0, 0, 0};
float errorRollPitch[3] = {0, 0, 0};
float errorYaw[3] = {0, 0, 0};
float DCM_Matrix[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
float Update_Matrix[3][3] = {{0, 1, 2}, {3, 4, 5}, {6, 7, 8}};
float Temporary_Matrix[3][3] = {{0, 0, 0}, {0, 0, 0}, {0, 0, 0}};
```

```
// Euler angles
float yaw;
float pitch;
float roll;

// DCM timing in the main loop
unsigned long timestamp;
unsigned long timestamp_old;
float G_Dt; // Integration time for DCM algorithm

// More output-state variables
boolean output_stream_on;
boolean output_single_on;
int curr_calibration_sensor = 0;
boolean reset_calibration_session_flag = true;
int num_accel_errors = 0;
int num_magn_errors = 0;
int num_gyro_errors = 0;

void read_sensors() {
  Read_Gyro(); // Read gyroscope
  Read_Accel(); // Read accelerometer
  Read_Magn(); // Read magnetometer
}

// Read every sensor and record a time stamp
// Init DCM with unfiltered orientation
// TODO re-init global vars?
void reset_sensor_fusion() {
  float temp1[3];
  float temp2[3];
  float xAxis[] = {1.0f, 0.0f, 0.0f};

  read_sensors();
  timestamp = millis();

  // GET PITCH
  // Using y-z-plane-component/x-component of gravity vector
  pitch = -atan2(accel[0], sqrt(accel[1] * accel[1] + accel[2] * accel[2]));

  // GET ROLL
  // Compensate pitch of gravity vector
  Vector_Cross_Product(temp1, accel, xAxis);
  Vector_Cross_Product(temp2, xAxis, temp1);
  // Normally using x-z-plane-component/y-component of compensated gravity vector
  // roll = atan2(temp2[1], sqrt(temp2[0] * temp2[0] + temp2[2] * temp2[2]));
  // Since we compensated for pitch, x-z-plane-component equals z-component:
  roll = atan2(temp2[1], temp2[2]);

  // GET YAW
  Compass_Heading();
  yaw = MAG_Heading;

  // Init rotation matrix
  init_rotation_matrix(DCM_Matrix, yaw, pitch, roll);
}

// Apply calibration to raw sensor readings
void compensate_sensor_errors() {
    // Compensate accelerometer error
    accel[0] = (accel[0] - ACCEL_X_OFFSET) * ACCEL_X_SCALE;
    accel[1] = (accel[1] - ACCEL_Y_OFFSET) * ACCEL_Y_SCALE;
    accel[2] = (accel[2] - ACCEL_Z_OFFSET) * ACCEL_Z_SCALE;

    // Compensate magnetometer error
#if CALIBRATION__MAGN_USE_EXTENDED == true
```

```
    for (int i = 0; i < 3; i++)
      magnetom_tmp[i] = magnetom[i] - magn_ellipsoid_center[i];
    Matrix_Vector_Multiply(magn_ellipsoid_transform, magnetom_tmp, magnetom);
#else
    magnetom[0] = (magnetom[0] - MAGN_X_OFFSET) * MAGN_X_SCALE;
    magnetom[1] = (magnetom[1] - MAGN_Y_OFFSET) * MAGN_Y_SCALE;
    magnetom[2] = (magnetom[2] - MAGN_Z_OFFSET) * MAGN_Z_SCALE;
#endif

    // Compensate gyroscope error
    gyro[0] -= GYRO_AVERAGE_OFFSET_X;
    gyro[1] -= GYRO_AVERAGE_OFFSET_Y;
    gyro[2] -= GYRO_AVERAGE_OFFSET_Z;
}

// Reset calibration session if reset_calibration_session_flag is set
void check_reset_calibration_session()
{
  // Raw sensor values have to be read already, but no error compensation applied

  // Reset this calibration session?
  if (!reset_calibration_session_flag) return;

  // Reset acc and mag calibration variables
  for (int i = 0; i < 3; i++) {
    accel_min[i] = accel_max[i] = accel[i];
    magnetom_min[i] = magnetom_max[i] = magnetom[i];
  }

  // Reset gyro calibration variables
  gyro_num_samples = 0;  // Reset gyro calibration averaging
  gyro_average[0] = gyro_average[1] = gyro_average[2] = 0.0f;

  reset_calibration_session_flag = false;
}

void turn_output_stream_on()
{
  output_stream_on = true;
  digitalWrite(STATUS_LED_PIN, LOW);
}

void turn_output_stream_off()
{
  output_stream_on = false;
  digitalWrite(STATUS_LED_PIN, LOW);
}

// Blocks until another byte is available on serial port
char readChar()
{
  while (Serial.available() < 1) { } // Block
  return Serial.read();
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////
//////   SETUP ///////////

void setup()
{
  // Init serial output
  Serial.begin(OUTPUT__BAUD_RATE);

  myservo.attach(6, 1000, 2000);
  pinMode(BOOM_PIN, INPUT);
  pinMode(MASSPOS_PIN, INPUT);
  pinMode(AUTOC_PIN, INPUT);
  pinMode(HEELSET_PIN, INPUT);
  digitalWrite(AUTOC_PIN, LOW);



////////////////////////////////////////////////////////////////////
  // Init status LED
  pinMode (STATUS_LED_PIN, OUTPUT);
  digitalWrite(STATUS_LED_PIN, LOW);

  // Init sensors
  delay(50);  // Give sensors enough time to start
  I2C_Init();
  Accel_Init();
  Magn_Init();
  Gyro_Init();

  // Read sensors, init DCM algorithm
  delay(20);  // Give sensors enough time to collect data
  reset_sensor_fusion();

  // Init output
#if (OUTPUT__HAS_RN_BLUETOOTH == true) || (OUTPUT__STARTUP_STREAM_ON == false)
  turn_output_stream_off();
#else
  turn_output_stream_on();
#endif
}



int hval;    //heel value -20 to 20
int boomval;   //boom switch 0 or 1
int autoc;
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////////////

// Main loop
void loop()
{

autoc = digitalRead(AUTOC_PIN);
/*
if (autoc < 560)
  ac = 0;
else
  ac = 1;
*/
Serial.print("autoc=");
Serial.print(autoc); Serial.print(",");

  int PVAL;
  PVAL = analogRead(MASSPOS_PIN);     //Read in potentiometer for location of the mass on the track
  int pval = map(PVAL, 0, 1023, -50, 50);

int heelset = analogRead(HEELSET_PIN);     //Reads in the heel set dial
int hvalset = map(heelset, 0, 1023, -20, 20);  //Constrains the limits from -20 to 20


///initializes the boom val to 0 or 1 (for testing purposes without the boom sensor plugged in )


  boomval = analogRead(BOOM_PIN);
```

```
 if (boomval == 1){
   hval = -abs(hvalset);
 }
 else{
   hval = abs(hvalset);
 }




/////////////////////// SENSOR SMOOTHING FOR JITTER//////////////////////////////

 double sensVal = -TO_DEG(roll)+2;          // for raw sensor values
 float filterVal;       // this determines smoothness  - .0001 is max  1 is off (no smoothing)
 double smoothedVal = TO_DEG(roll);     // this holds the last loop value just use a unique variable for every different sensor that needs
smoothing
 int i, j;             // loop counters or demo


   filterVal = .2 * .15;

     // sensVal = analogRead(0);   this is what one would do normally

 smoothedVal =  smooth(sensVal, filterVal, smoothedVal);   // second parameter determines smoothness  - 0 is off,  .9999 is max smooth

 double rollDegrees = smoothedVal;


 Serial.print("angin=");
 Serial.print(rollDegrees); Serial.print(",");
 Serial.print("hset=");
 Serial.print(hval); Serial.print(",");


/////////////////////////// PID CALLS/////////////////

int gapDist = 2;

//Aggressive Reaction
double aggK=1, aggKp=5, aggKi=0, aggKd=0;
//Conservative Reactions
double consK = 1, consKp= 3, consKi=0, consKd=0;

double gap = abs(hval-sensVal); //distance away from setpoint

int x;

 if (gap<gapDist)
 { //we're close to setpoint, use conservative tuning parameters
   x = updatePid(hval, sensVal, consK, consKp, consKi, consKd);
 //   int pwm = map(drive, 127, 255)
 }
 else{
   //we're far from setpoint, use aggressive tuning parameters
   x= updatePid(hval, sensVal, aggK, aggKp, aggKi, aggKd);
 }

 int y = updatePid(0, pval, 1, 3, 0, 0);      //pid for the auto center button and stop


Serial.print("pidoutx=");
Serial.print(x); Serial.print(",");

Serial.print("pin=");
Serial.print(pval); Serial.print(",");
Serial.print("set=");
Serial.print(0); Serial.print(",");
Serial.print("pidouty=");
Serial.print(y); Serial.print(",");
/////////////////////////////////////////////// AUTO CENTER BUTTON /////////////////////////
int z= ((36-pval)/72)*180;


  if (autoc == HIGH){      //checks to see if the auto center button is high
    if(pval==0)
       myservo.write(90);   //stops the motor
    else
      myservo.write(90+y);  //centers the motor based on the mass position PID
  }
  else {
   if ( pval < - 45 || pval > 45 || sensVal < -30 || sensVal > 30  ) //end of track & max heel check
      myservo.write(90+x);   //moves the motor in the opposite direction
   else
      myservo.write(90-x);   //centers the motor at the heel set angle based on the measured heel PID
  }

  int autocoff = 90-x;
  int autocon =  90-y;



/*
 int range = map(angerr, 0, 20, 0, 3);

 // speed of the motor depending on the
 // difference value of the angle error(faster speed for a higher delta change)
 /* switch (range) {
 case 0:    // slow
   pwm(SLOW);
   break;
 case 1:    // medium
   pwm(MEDIUM);
   break;
 case 2:    // fast
   pwm(FAST);
   break;
 case 3:    // full
   pwm(FULL);
   break;
 }


 */
 // Read incoming control messages
 if (Serial.available() >= 2)
 {
   if (Serial.read() == '#') // Start of new control message
   {
     int command = Serial.read(); // Commands
     if (command == 'f') // request one output _f_rame
       output_single_on = true;
     else if (command == 's') // _s_ynch request
     {
       // Read ID
       byte id[2];
       id[0] = readChar();
       id[1] = readChar();

       // Reply with synch message
       Serial.print("#SYNCH");
       Serial.write(id, 2);
       Serial.println();
     }
     else if (command == 'o') // Set _o_utput mode
     {
       char output_param = readChar();
       if (output_param == 'n')  // Calibrate _n_ext sensor
       {
         curr_calibration_sensor = (curr_calibration_sensor + 1) % 3;
         reset_calibration_session_flag = true;
       }
       else if (output_param == 't') // Output angles as _t_ext
       {
         output_mode = OUTPUT__MODE_ANGLES;
         output_format = OUTPUT__FORMAT_TEXT;
       }
       else if (output_param == 'b') // Output angles in _b_inary format
       {
         output_mode = OUTPUT__MODE_ANGLES;
         output_format = OUTPUT__FORMAT_BINARY;
       }
       else if (output_param == 'c') // Go to _c_alibration mode
       {
         output_mode = OUTPUT__MODE_CALIBRATE_SENSORS;
         reset_calibration_session_flag = true;
       }
       else if (output_param == 's') // Output _s_ensor values
       {
         char values_param = readChar();
         char format_param = readChar();
         if (values_param == 'r')  // Output _r_aw sensor values
           output_mode = OUTPUT__MODE_SENSORS_RAW;
         else if (values_param == 'c')  // Output _c_alibrated sensor values
           output_mode = OUTPUT__MODE_SENSORS_CALIB;
         else if (values_param == 'b')  // Output _b_oth sensor values (raw and calibrated)
           output_mode = OUTPUT__MODE_SENSORS_BOTH;

         if (format_param == 't') // Output values as _t_text
           output_format = OUTPUT__FORMAT_TEXT;
         else if (format_param == 'b') // Output values in _b_inary format
           output_format = OUTPUT__FORMAT_BINARY;
       }
       else if (output_param == '0') // Disable continuous streaming output
       {
         turn_output_stream_off();
         reset_calibration_session_flag = true;
       }
       else if (output_param == '1') // Enable continuous streaming output
       {
         reset_calibration_session_flag = true;
         turn_output_stream_on();
       }
       else if (output_param == 'e') // _e_rror output settings
       {
         char error_param = readChar();
         if (error_param == '0') output_errors = false;
         else if (error_param == '1') output_errors = true;
         else if (error_param == 'c') // get error count
         {
           Serial.print("#AMG-ERR:");
           Serial.print(num_accel_errors); Serial.print(",");
           Serial.print(num_magn_errors); Serial.print(",");
           Serial.println(num_gyro_errors);
         }
       }
     }
#if OUTPUT__HAS_RN_BLUETOOTH == true
     // Read messages from bluetooth module
```

```
      // For this to work, the connect/disconnect message prefix of the module has to be set to "#".
      else if (command == 'C') // Bluetooth "#CONNECT" message (does the same as "#o1")
        turn_output_stream_on();
      else if (command == 'D') // Bluetooth "#DISCONNECT" message (does the same as "#o0")
        turn_output_stream_off();
#endif // OUTPUT__HAS_RN_BLUETOOTH == true
    }
    else
    { } // Skip character
  }

  // Time to read the sensors again?
  if((millis() - timestamp) >= OUTPUT__DATA_INTERVAL)
  {
    timestamp_old = timestamp;
    timestamp = millis();
    if (timestamp > timestamp_old)
      G_Dt = (float) (timestamp - timestamp_old) / 1000.0f; // Real time of loop run. We use this on the DCM algorithm (gyro integration time)
    else G_Dt = 0;

    // Update sensor readings
    read_sensors();

    if (output_mode == OUTPUT__MODE_CALIBRATE_SENSORS)  // We're in calibration mode
    {
      check_reset_calibration_session();  // Check if this session needs a reset
      if (output_stream_on || output_single_on) output_calibration(curr_calibration_sensor);
    }
    else if (output_mode == OUTPUT__MODE_ANGLES)  // Output angles
    {
      // Apply sensor calibration
      compensate_sensor_errors();

      // Run DCM algorithm
      Compass_Heading(); // Calculate magnetic heading
      Matrix_update();
      Normalize();
      Drift_correction();
      Euler_angles();

      if (output_stream_on || output_single_on) output_angles();
    }
    else  // Output sensor values
    {
      if (output_stream_on || output_single_on) output_sensors();
    }

    output_single_on = false;

#if DEBUG__PRINT_LOOP_TIME == true
    Serial.print("loop time (ms) = ");
    Serial.println(millis() - timestamp);
#endif
  }
#if DEBUG__PRINT_LOOP_TIME == true
  else
  {
    Serial.println("waiting...");
  }
#endif
}

/* This file is part of the Razor AHRS Firmware */

void Compass_Heading()
{
  float mag_x;
  float mag_y;
  float cos_roll;
  float sin_roll;
  float cos_pitch;
  float sin_pitch;

  cos_roll = cos(roll);
  sin_roll = sin(roll);
  cos_pitch = cos(pitch);
  sin_pitch = sin(pitch);

  // Tilt compensated magnetic field X
  mag_x = magnetom[0] * cos_pitch + magnetom[1] * sin_roll * sin_pitch + magnetom[2] * cos_roll * sin_pitch;
  // Tilt compensated magnetic field Y
  mag_y = magnetom[1] * cos_roll - magnetom[2] * sin_roll;
  // Magnetic Heading
  MAG_Heading = atan2(-mag_y, mag_x);
}

/* This file is part of the Razor AHRS Firmware */

// DCM algorithm

/**************************************************/
void Normalize(void)
{
  float error=0;
  float temporary[3][3];
  float renorm=0;

  error= -Vector_Dot_Product(&DCM_Matrix[0][0],&DCM_Matrix[1][0])*.5; //eq.19

  Vector_Scale(&temporary[0][0], &DCM_Matrix[1][0], error); //eq.19
  Vector_Scale(&temporary[1][0], &DCM_Matrix[0][0], error); //eq.19

  Vector_Add(&temporary[0][0], &temporary[0][0], &DCM_Matrix[0][0]);//eq.19
  Vector_Add(&temporary[1][0], &temporary[1][0], &DCM_Matrix[1][0]);//eq.19

  Vector_Cross_Product(&temporary[2][0],&temporary[0][0],&temporary[1][0]); // c= a x b //eq.20

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[0][0],&temporary[0][0])); //eq.21
  Vector_Scale(&DCM_Matrix[0][0], &temporary[0][0], renorm);

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[1][0],&temporary[1][0])); //eq.21
  Vector_Scale(&DCM_Matrix[1][0], &temporary[1][0], renorm);

  renorm= .5 *(3 - Vector_Dot_Product(&temporary[2][0],&temporary[2][0])); //eq.21
  Vector_Scale(&DCM_Matrix[2][0], &temporary[2][0], renorm);
}

/**************************************************/
void Drift_correction(void)
{
  float mag_heading_x;
  float mag_heading_y;
  float errorCourse;
  //Compensation the Roll, Pitch and Yaw drift.
  static float Scaled_Omega_P[3];
  static float Scaled_Omega_I[3];
  float Accel_magnitude;
  float Accel_weight;


  //*****Roll and Pitch***************
  // Calculate the magnitude of the accelerometer vector
  Accel_magnitude = sqrt(Accel_Vector[0]*Accel_Vector[0] + Accel_Vector[1]*Accel_Vector[1] + Accel_Vector[2]*Accel_Vector[2]);
  Accel_magnitude = Accel_magnitude / GRAVITY; // Scale to gravity.
  // Dynamic weighting of accelerometer info (reliability filter)
  // Weight for accelerometer info (<0.5G = 0.0, 1G = 1.0 , >1.5G = 0.0)
  Accel_weight = constrain(1 - 2*abs(1 - Accel_magnitude),0,1);  //

  Vector_Cross_Product(&errorRollPitch[0],&Accel_Vector[0],&DCM_Matrix[2][0]); //adjust the ground of reference
  Vector_Scale(&Omega_P[0],&errorRollPitch[0],Kp_ROLLPITCH*Accel_weight);

  Vector_Scale(&Scaled_Omega_I[0],&errorRollPitch[0],Ki_ROLLPITCH*Accel_weight);
  Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);

  //*****YAW***************
  // We make the gyro YAW drift correction based on compass magnetic heading

  mag_heading_x = cos(MAG_Heading);
  mag_heading_y = sin(MAG_Heading);
  errorCourse=(DCM_Matrix[0][0]*mag_heading_y) - (DCM_Matrix[1][0]*mag_heading_x);  //Calculating YAW error
  Vector_Scale(errorYaw,&DCM_Matrix[2][0],errorCourse); //Applys the yaw correction to the XYZ rotation of the aircraft, depeding the
position.

  Vector_Scale(&Scaled_Omega_P[0],&errorYaw[0],Kp_YAW);//.01proportional of YAW.
  Vector_Add(Omega_P,Omega_P,Scaled_Omega_P);//Adding  Proportional.

  Vector_Scale(&Scaled_Omega_I[0],&errorYaw[0],Ki_YAW);//.00001Integrator
  Vector_Add(Omega_I,Omega_I,Scaled_Omega_I);//adding integrator to the Omega_I
}

void Matrix_update(void)
{
  Gyro_Vector[0]=GYRO_SCALED_RAD(gyro[0]); //gyro x roll
  Gyro_Vector[1]=GYRO_SCALED_RAD(gyro[1]); //gyro y pitch
  Gyro_Vector[2]=GYRO_SCALED_RAD(gyro[2]); //gyro z yaw

  Accel_Vector[0]=accel[0];
  Accel_Vector[1]=accel[1];
  Accel_Vector[2]=accel[2];

  Vector_Add(&Omega[0], &Gyro_Vector[0], &Omega_I[0]);  //adding proportional term
  Vector_Add(&Omega_Vector[0], &Omega[0], &Omega_P[0]); //adding Integrator term

#if DEBUG__NO_DRIFT_CORRECTION == true // Do not use drift correction
  Update_Matrix[0][0]=0;
  Update_Matrix[0][1]=-G_Dt*Gyro_Vector[2];//-z
  Update_Matrix[0][2]=G_Dt*Gyro_Vector[1];//y
  Update_Matrix[1][0]=G_Dt*Gyro_Vector[2];//z
  Update_Matrix[1][1]=0;
  Update_Matrix[1][2]=-G_Dt*Gyro_Vector[0];
  Update_Matrix[2][0]=-G_Dt*Gyro_Vector[1];
  Update_Matrix[2][1]=G_Dt*Gyro_Vector[0];
  Update_Matrix[2][2]=0;
#else // Use drift correction
  Update_Matrix[0][0]=0;
  Update_Matrix[0][1]=-G_Dt*Omega_Vector[2];//-z
  Update_Matrix[0][2]=G_Dt*Omega_Vector[1];//y
  Update_Matrix[1][0]=G_Dt*Omega_Vector[2];//z
  Update_Matrix[1][1]=0;
  Update_Matrix[1][2]=-G_Dt*Omega_Vector[0];//-x
  Update_Matrix[2][0]=-G_Dt*Omega_Vector[1];//-y
  Update_Matrix[2][1]=G_Dt*Omega_Vector[0];//x
  Update_Matrix[2][2]=0;
#endif

  Matrix_Multiply(DCM_Matrix,Update_Matrix,Temporary_Matrix); //a*b=c
```

```
 for(int x=0; x<3; x++) //Matrix Addition (update)
 {
   for(int y=0; y<3; y++)
   {
     DCM_Matrix[x][y]+=Temporary_Matrix[x][y];
   }
 }
}

void Euler_angles(void)
{
 pitch = -asin(DCM_Matrix[2][0]);
 roll = atan2(DCM_Matrix[2][1],DCM_Matrix[2][2]);
 yaw = atan2(DCM_Matrix[1][0],DCM_Matrix[0][0]);
}

 int smooth(int data, float filterVal, float smoothedVal){

   if (filterVal > 1){      // check to make sure param's are within range
     filterVal = .99;
   }
   else if (filterVal <= 0){
     filterVal = 0;
   }

   //feeds back the output into the input
   smoothedVal = (data * (1 - filterVal)) + (smoothedVal  *  filterVal);

   return (int)smoothedVal;
 }


   float Q_angle  =  0.001; //0.001
   float Q_gyro   =  0.003;  //0.003
   float R_angle  =  0.03;  //0.03

   float x_angle = 0;
   float x_bias = 0;
   float P_00 = 0, P_01 = 0, P_10 = 0, P_11 = 0;
   float dt, y, S;
   float K_0, K_1;

 float kalmanCalculate(float newAngle, float newRate,int looptime) {
   dt = float(looptime)/1000;
   x_angle += dt * (newRate - x_bias);
   P_00 +=  - dt * (P_10 + P_01) + Q_angle * dt;
   P_01 +=  - dt * P_11;
   P_10 +=  - dt * P_11;
   P_11 +=  + Q_gyro * dt;

   y = newAngle - x_angle;
   S = P_00 + R_angle;
   K_0 = P_00 / S;
   K_1 = P_10 / S;

   x_angle +=  K_0 * y;
   x_bias  +=  K_1 * y;
   P_00 -= K_0 * P_00;
   P_01 -= K_0 * P_01;
   P_10 -= K_1 * P_00;
   P_11 -= K_1 * P_01;

   return x_angle;
 }

/* This file is part of the Razor AHRS Firmware */
```

```
// Computes the dot product of two vectors
float Vector_Dot_Product(const float v1[3], const float v2[3])
{
  float result = 0;

  for(int c = 0; c < 3; c++)
  {
    result += v1[c] * v2[c];
  }

  return result;
}

// Computes the cross product of two vectors
// out has to different from v1 and v2 (no in-place)!
void Vector_Cross_Product(float out[3], const float v1[3], const float v2[3])
{
  out[0] = (v1[1] * v2[2]) - (v1[2] * v2[1]);
  out[1] = (v1[2] * v2[0]) - (v1[0] * v2[2]);
  out[2] = (v1[0] * v2[1]) - (v1[1] * v2[0]);
}

// Multiply the vector by a scalar
void Vector_Scale(float out[3], const float v[3], float scale)
{
  for(int c = 0; c < 3; c++)
  {
    out[c] = v[c] * scale;
  }
}

// Adds two vectors
void Vector_Add(float out[3], const float v1[3], const float v2[3])
{
  for(int c = 0; c < 3; c++)
  {
    out[c] = v1[c] + v2[c];
  }
}

// Multiply two 3x3 matrices: out = a * b
// out has to different from a and b (no in-place)!
void Matrix_Multiply(const float a[3][3], const float b[3][3], float out[3][3])
{
  for(int x = 0; x < 3; x++)  // rows
  {
    for(int y = 0; y < 3; y++)  // columns
    {
      out[x][y] = a[x][0] * b[0][y] + a[x][1] * b[1][y] + a[x][2] * b[2][y];
    }
  }
}

// Multiply 3x3 matrix with vector: out = a * b
// out has to different from b (no in-place)!
void Matrix_Vector_Multiply(const float a[3][3], const float b[3], float out[3])
{
  for(int x = 0; x < 3; x++)
  {
    out[x] = a[x][0] * b[0] + a[x][1] * b[1] + a[x][2] * b[2];
  }
}

// Init rotation matrix using euler angles
void init_rotation_matrix(float m[3][3], float yaw, float pitch, float roll)
```

```
{
  float c1 = cos(roll);
  float s1 = sin(roll);
  float c2 = cos(pitch);
  float s2 = sin(pitch);
  float c3 = cos(yaw);
  float s3 = sin(yaw);

  // Euler angles, right-handed, intrinsic, XYZ convention
  // (which means: rotate around body axes Z, Y', X'')
  m[0][0] = c2 * c3;
  m[0][1] = c3 * s1 * s2 - c1 * s3;
  m[0][2] = s1 * s3 + c1 * c3 * s2;

  m[1][0] = c2 * s3;
  m[1][1] = c1 * c3 + s1 * s2 * s3;
  m[1][2] = c1 * s2 * s3 - c3 * s1;

  m[2][0] = -s2;
  m[2][1] = c2 * s1;
  m[2][2] = c1 * c2;
}

/* This file is part of the Razor AHRS Firmware */

// Output angles: yaw, pitch, roll


void output_angles()
{

  if (output_format == OUTPUT__FORMAT_BINARY)
  {
    float ypr[3];
    ypr[0] = TO_DEG(yaw);
    ypr[1] = TO_DEG(pitch);
    ypr[2] = TO_DEG(roll);
    Serial.write((byte*) ypr, 12);  // No new-line
  }
  else if (output_format == OUTPUT__FORMAT_TEXT)
  {
    Serial.print("#YPR=");
    Serial.print(TO_DEG(yaw)); Serial.print(",");
    Serial.print(TO_DEG(pitch)); Serial.print(",");
    Serial.print(TO_DEG(roll)); Serial.print(",");
    Serial.println();
    delay(30);
  }
}

void output_calibration(int calibration_sensor)
{
  if (calibration_sensor == 0)  // Accelerometer
  {
    // Output MIN/MAX values
    Serial.print("accel x,y,z (min/max) = ");
    for (int i = 0; i < 3; i++) {
      if (accel[i] < accel_min[i]) accel_min[i] = accel[i];
      if (accel[i] > accel_max[i]) accel_max[i] = accel[i];
      Serial.print(accel_min[i]);
      Serial.print("/");
      Serial.print(accel_max[i]);
      if (i < 2) Serial.print("  ");
      else Serial.println();
    }
  }
  else if (calibration_sensor == 1)  // Magnetometer
  {
    // Output MIN/MAX values
    Serial.print("magn x,y,z (min/max) = ");
    for (int i = 0; i < 3; i++) {
      if (magnetom[i] < magnetom_min[i]) magnetom_min[i] = magnetom[i];
      if (magnetom[i] > magnetom_max[i]) magnetom_max[i] = magnetom[i];
      Serial.print(magnetom_min[i]);
      Serial.print("/");
      Serial.print(magnetom_max[i]);
      if (i < 2) Serial.print("  ");
      else Serial.println();
    }
  }
  else if (calibration_sensor == 2)  // Gyroscope
  {
    // Average gyro values
    for (int i = 0; i < 3; i++)
      gyro_average[i] += gyro[i];
    gyro_num_samples++;

    // Output current and averaged gyroscope values
    Serial.print("gyro x,y,z (current/average) = ");
    for (int i = 0; i < 3; i++) {
      Serial.print(gyro[i]);
      Serial.print("/");
      Serial.print(gyro_average[i] / (float) gyro_num_samples);
      if (i < 2) Serial.print("  ");
      else Serial.println();
    }
  }
}

void output_sensors_text(char raw_or_calibrated)
{
  Serial.print("#A-"); Serial.print(raw_or_calibrated); Serial.print('=');
  Serial.print(accel[0]); Serial.print(",");
  Serial.print(accel[1]); Serial.print(",");
  Serial.print(accel[2]); Serial.println();

  Serial.print("#M-"); Serial.print(raw_or_calibrated); Serial.print('=');
  Serial.print(magnetom[0]); Serial.print(",");
  Serial.print(magnetom[1]); Serial.print(",");
  Serial.print(magnetom[2]); Serial.println();

  Serial.print("#G-"); Serial.print(raw_or_calibrated); Serial.print('=');
  Serial.print(gyro[0]); Serial.print(",");
  Serial.print(gyro[1]); Serial.print(",");
  Serial.print(gyro[2]); Serial.println();
}

void output_sensors_binary()
{
  Serial.write((byte*) accel, 12);
  Serial.write((byte*) magnetom, 12);
  Serial.write((byte*) gyro, 12);
}

void output_sensors()
{
  if (output_mode == OUTPUT__MODE_SENSORS_RAW)
  {
    if (output_format == OUTPUT__FORMAT_BINARY)
      output_sensors_binary();
    else if (output_format == OUTPUT__FORMAT_TEXT)
      output_sensors_text('R');
```

```
 }
 else if (output_mode == OUTPUT__MODE_SENSORS_CALIB)
 {
   // Apply sensor calibration
   compensate_sensor_errors();

   if (output_format == OUTPUT__FORMAT_BINARY)
     output_sensors_binary();
   else if (output_format == OUTPUT__FORMAT_TEXT)
     output_sensors_text('C');
 }
 else if (output_mode == OUTPUT__MODE_SENSORS_BOTH)
 {
   if (output_format == OUTPUT__FORMAT_BINARY)
   {
     output_sensors_binary();
     compensate_sensor_errors();
     output_sensors_binary();
   }
   else if (output_format == OUTPUT__FORMAT_TEXT)
   {
     output_sensors_text('R');
     compensate_sensor_errors();
     output_sensors_text('C');
   }
 }
}

#include <math.h>

unsigned long lastTime;
int last_error = 0;
int integrated_error = 0;
int pTerm = 0, iTerm = 0, dTerm = 0;
int error;

int updatePid(int targetPosition, int currentPosition, float K, int Kp, int Ki, int Kd) {


 /*How long since we last calculated*/
  unsigned long now = millis();
  double timeChange = (double)(now - lastTime);

 error = targetPosition - currentPosition;

 integrated_error += (error*timeChange);

 if (abs(error) < 2)
         error = 0;

 pTerm = Kp * error;

 iTerm = Ki * integrated_error;
 dTerm = Kd * (error - last_error)/timeChange;

 last_error = error;
 lastTime = now;


 return  constrain(K*(pTerm + iTerm + dTerm), -100, 100 );




}
```

```
/* This file is part of the Razor AHRS Firmware */

// I2C code to read the sensors

// Sensor I2C addresses
#define ACCEL_ADDRESS ((int) 0x53) // 0x53 = 0xA6 / 2
#define MAGN_ADDRESS  ((int) 0x1E) // 0x1E = 0x3C / 2
#define GYRO_ADDRESS  ((int) 0x68) // 0x68 = 0xD0 / 2

// Arduino backward compatibility macros
#if ARDUINO >= 100
  #define WIRE_SEND(b) Wire.write((byte) b)
  #define WIRE_RECEIVE() Wire.read()
#else
  #define WIRE_SEND(b) Wire.send(b)
  #define WIRE_RECEIVE() Wire.receive()
#endif


void I2C_Init()
{
  Wire.begin();
}

void Accel_Init()
{
  Wire.beginTransmission(ACCEL_ADDRESS);
  WIRE_SEND(0x2D);  // Power register
  WIRE_SEND(0x08);  // Measurement mode
  Wire.endTransmission();
  delay(5);
  Wire.beginTransmission(ACCEL_ADDRESS);
  WIRE_SEND(0x31);  // Data format register
  WIRE_SEND(0x08);  // Set to full resolution
  Wire.endTransmission();
  delay(5);

  // Because our main loop runs at 50Hz we adjust the output data rate to 50Hz (25Hz bandwidth)
  Wire.beginTransmission(ACCEL_ADDRESS);
  WIRE_SEND(0x2C);  // Rate
  WIRE_SEND(0x09);  // Set to 50Hz, normal operation
  Wire.endTransmission();
  delay(5);
}

// Reads x, y and z accelerometer registers
void Read_Accel()
{
  int i = 0;
  byte buff[6];

  Wire.beginTransmission(ACCEL_ADDRESS);
  WIRE_SEND(0x32);  // Send address to read from
  Wire.endTransmission();

  Wire.beginTransmission(ACCEL_ADDRESS);
  Wire.requestFrom(ACCEL_ADDRESS, 6);  // Request 6 bytes
  while(Wire.available())  // ((Wire.available())&&(i<6))
  {
    buff[i] = WIRE_RECEIVE();  // Read one byte
    i++;
  }
  Wire.endTransmission();

  if (i == 6)  // All bytes received?
  {
    // No multiply by -1 for coordinate system transformation here, because of double negation:
    // We want the gravity vector, which is negated acceleration vector.
    accel[0] = (((int) buff[3]) << 8) | buff[2];  // X axis (internal sensor y axis)
    accel[1] = (((int) buff[1]) << 8) | buff[0];  // Y axis (internal sensor x axis)
    accel[2] = (((int) buff[5]) << 8) | buff[4];  // Z axis (internal sensor z axis)
  }
  else
  {
    num_accel_errors++;
    if (output_errors) Serial.println("!ERR: reading accelerometer");
  }
}

void Magn_Init()
{
  Wire.beginTransmission(MAGN_ADDRESS);
  WIRE_SEND(0x02);
  WIRE_SEND(0x00);  // Set continuous mode (default 10Hz)
  Wire.endTransmission();
  delay(5);

  Wire.beginTransmission(MAGN_ADDRESS);
  WIRE_SEND(0x00);
  WIRE_SEND(0b00011000);  // Set 50Hz
  Wire.endTransmission();
  delay(5);
}

void Read_Magn()
{
  int i = 0;
  byte buff[6];

  Wire.beginTransmission(MAGN_ADDRESS);
  WIRE_SEND(0x03);  // Send address to read from
  Wire.endTransmission();

  Wire.beginTransmission(MAGN_ADDRESS);
  Wire.requestFrom(MAGN_ADDRESS, 6);  // Request 6 bytes
  while(Wire.available())  // ((Wire.available())&&(i<6))
  {
    buff[i] = WIRE_RECEIVE();  // Read one byte
    i++;
  }
  Wire.endTransmission();

  if (i == 6)  // All bytes received?
  {
// 9DOF Razor IMU SEN-10125 using HMC5843 magnetometer
#if HW__VERSION_CODE == 10125
    // MSB byte first, then LSB; X, Y, Z
    magnetom[0] = -1 * ((((int) buff[2]) << 8) | buff[3]);  // X axis (internal sensor -y axis)
    magnetom[1] = -1 * ((((int) buff[0]) << 8) | buff[1]);  // Y axis (internal sensor -x axis)
    magnetom[2] = -1 * ((((int) buff[4]) << 8) | buff[5]);  // Z axis (internal sensor -z axis)
// 9DOF Razor IMU SEN-10736 using HMC5883L magnetometer
#elif HW__VERSION_CODE == 10736
    // MSB byte first, then LSB; Y and Z reversed: X, Z, Y
    magnetom[0] = -1 * ((((int) buff[4]) << 8) | buff[5]);  // X axis (internal sensor -y axis)
    magnetom[1] = -1 * ((((int) buff[0]) << 8) | buff[1]);  // Y axis (internal sensor -x axis)
    magnetom[2] = -1 * ((((int) buff[2]) << 8) | buff[3]);  // Z axis (internal sensor -z axis)
// 9DOF Sensor Stick SEN-10183 and SEN-10321 using HMC5843 magnetometer
#elif (HW__VERSION_CODE == 10183) || (HW__VERSION_CODE == 10321)
    // MSB byte first, then LSB; X, Y, Z
    magnetom[0] = (((int) buff[0]) << 8) | buff[1];         // X axis (internal sensor x axis)
    magnetom[1] = -1 * ((((int) buff[2]) << 8) | buff[3]);  // Y axis (internal sensor -y axis)
    magnetom[2] = -1 * ((((int) buff[4]) << 8) | buff[5]);  // Z axis (internal sensor -z axis)
```

```
// 9DOF Sensor Stick SEN-10724 using HMC5883L magnetometer
#elif HW__VERSION_CODE == 10724
    // MSB byte first, then LSB; Y and Z reversed: X, Z, Y
    magnetom[0] = (((int) buff[0]) << 8) | buff[1];         // X axis (internal sensor x axis)
    magnetom[1] = -1 * ((((int) buff[4]) << 8) | buff[5]);  // Y axis (internal sensor -y axis)
    magnetom[2] = -1 * ((((int) buff[2]) << 8) | buff[3]);  // Z axis (internal sensor -z axis)
#endif
  }
  else
  {
    num_magn_errors++;
    if (output_errors) Serial.println("!ERR: reading magnetometer");
  }
}

void Gyro_Init()
{
  // Power up reset defaults
  Wire.beginTransmission(GYRO_ADDRESS);
  WIRE_SEND(0x3E);
  WIRE_SEND(0x80);
  Wire.endTransmission();
  delay(5);

  // Select full-scale range of the gyro sensors
  // Set LP filter bandwidth to 42Hz
  Wire.beginTransmission(GYRO_ADDRESS);
  WIRE_SEND(0x16);
  WIRE_SEND(0x1B);  // DLPF_CFG = 3, FS_SEL = 3
  Wire.endTransmission();
  delay(5);

  // Set sample rato to 50Hz
  Wire.beginTransmission(GYRO_ADDRESS);
  WIRE_SEND(0x15);
  WIRE_SEND(0x0A);  //  SMPLRT_DIV = 10 (50Hz)
  Wire.endTransmission();
  delay(5);

  // Set clock to PLL with z gyro reference
  Wire.beginTransmission(GYRO_ADDRESS);
  WIRE_SEND(0x3E);
  WIRE_SEND(0x00);
  Wire.endTransmission();
  delay(5);
}

// Reads x, y and z gyroscope registers
void Read_Gyro()
{
  int i = 0;
  byte buff[6];

  Wire.beginTransmission(GYRO_ADDRESS);
  WIRE_SEND(0x1D);  // Sends address to read from
  Wire.endTransmission();

  Wire.beginTransmission(GYRO_ADDRESS);
  Wire.requestFrom(GYRO_ADDRESS, 6);  // Request 6 bytes
  while(Wire.available())  // ((Wire.available())&&(i<6))
  {
    buff[i] = WIRE_RECEIVE();  // Read one byte
    i++;
  }
  Wire.endTransmission();
  if (i == 6)  // All bytes received?
  {
    gyro[0] = -1 * ((((int) buff[2]) << 8) | buff[3]);    // X axis (internal sensor -y axis)
    gyro[1] = -1 * ((((int) buff[0]) << 8) | buff[1]);    // Y axis (internal sensor -x axis)
    gyro[2] = -1 * ((((int) buff[4]) << 8) | buff[5]);    // Z axis (internal sensor -z axis)
  }
  else
  {
    num_gyro_errors++;
    if (output_errors) Serial.println("!ERR: reading gyroscope");
  }
}
```

Appendix Q

Part	**Cost**
6061 Aluminum	$300
CIM Motor	$30
Talon Motor Controller	$60
TB3 Tough Box Gearbox	$200
Duracell Battery	$210
Sealed Bearings	$150
IMU PCB	$115
Various Parts (wires, hardware, etc.)	$300
Total	$1365

Bibliography

Alciatore, D. G., & Histand, M. B. (2007). Introduction to mechatronics and measurement systems: McGraw-Hill New York.

Alper, S. E., Temiz, Y., & Akin, T. (2008). A compact angular rate sensor system using a fully decoupled silicon-on-glass MEMS gyroscope. Microelectromechanical Systems, Journal of, 17(6), 1418-1429.

Bartz, P. (2013). Building an AHRS using the SparkFun "9DOF Razor IMU" or "9DOF Sensor Stick. Retrieved November 16, 2013, from https://github.com/ptrbrtz/razor-9dof-ahrs/wiki/Tutorial

Bass, D. W. (1998). Roll stabilization for small fishing vessels using paravanes and anti-roll tanks. Marine technology, 35(2), 74-84.

Browning, R. (2012). Flying Junior. Retrieved November 5, 2013, from http://sailboatdata.com/viewrecord.asp?class_id=3087

Cape Horn Engineering. (2013). Sail force coefficients. Retrieved November 2, 2013, from http://www.cape-horn-eng.com/sail-force-coefficients.html

Encyclopedia Britannica. (2013). Coriolis Effect. Retrieved November 15,, 2013, from http://www.britannica.com/EBchecked/topic/137646/Coriolis-force

Esfandyari, J., De Nuccio, R., & Xu, G. (2012). Introduction to MEMS gryoscopes. Retrieved from http://electroiq.com/blog/2010/11/introduction-to-mems-gyroscopes/

Future Electronics. (2013). What is an Angular Position Sensor? Retrieved November 13, 2013, from http://www.futureelectronics.com/en/sensors/angular-position.aspx

International. (2012). Robot Sport: Heavy Hitters. Retrieved April 28, 2014 from http://www.economist.com/node/21559391

International Yacht Racing Union. International FJ Class Rules. Retrieved November 6, 2013, from http://fjclassita.tripod.com/ruleseng.htm#sails

Kasten, M. (2012). An Overview of a Few Common Roll Attenuation Strategies For Motor Yachts and Motor Sailers. Retrieved November 14, 2013, from http://www.kastenmarine.com/roll_attenuation.htm

Kimball, J. (2010). Physics of Sailing. FL. USA: CRC Press.

Lewis, E. V. (1988). Principles of Naval Architecture Second Revision (Vol. I). NJ. USA: The Society of Naval Architects and Marine Engineers

Lewis, E. V. (1988a). Principles of Naval Architecture Second Revision (Vol. II). NJ.USA: The Society of Naval Architects and Marine Engineers.

Lewis, E. V. (1988b). Principles of Naval Architecture Second Revision (Vol. III). NJ. USA: The Society of Naval Architects and Marine Engineers.

Lochhaas, T. (2013). Learn How to Sail a Small Boat. Retrieved November 22, 2013, from http://sailing.about.com/od/sailasmallsailboat/ss/Learn-How-To-Sail-A-Small-Sailboat-3-Basic-Sailing-Techniques.htm

Michalak, J. (2003). Sail Area Math. Retrieved from http://www.boatbuilding-links.de/Jim-Michalak/sail-math-and-thoughts-of.html

Neely Chaulk &amp; Associates. (2010). Anti-Roll Tank. Retrieved November 9, 2013, from http://www.neely-chaulk.com/narciki/Anti-roll_tank

Power House Museum. (2002). Ethical and Social Implications. Retrieved April 20, 2014, from http://www.powerhousemuseum.com/hsc/aibo/ethics.htm

Royce, P. M. (1993). <i>Royce's sailing illustrated: the best of all sailing worlds</i> (Special sail course ed.). Newport Beach, CA: Royce Publications.

Sheng, L., Liang, F., Gao-yun, L., & Bing, L. (2008, 2008). *Fin/flap fin joint control for ship anti-roll system.* Paper presented at the Mechatronics and Automation, IEEE International Conference on.

STMicroeelectronics. (2011). Everything about STMicroelectronics' 3-axis digital MEMS gyroscopes. STMicroelectronics, Journal of, 1-40.

Townsend, N. C., Murphy, A. J., & Shenoi, R. A. (2007). A new active gyrostabiliser system for ride control of marine vehicles. *Ocean Engineering, 34*(11–12), 1607-1617. doi: http://dx.doi.org/10.1016/j.oceaneng.2006.11.004

Wikipedia. (2013). Ship Stability. Retrieved November 1, 2013, from http://en.wikipedia.org/wiki/Ship_stability#Active_systems

Wyatt, M. (2010). FJ Tuning Guide. Retrieved November 7, 2013, from http://www.westcoastsailing.net/media/pdf/laserperformance/FJTuningGuide.pdf

Youssef, K. S., Mook, D. T., Nayfeh, A. H., & Ragab, S. A. (2003). Roll stabilization by passive anti-roll tanks using an improved model of the tank-liquid motion. Journal of Vibration and Control, 9(7), 839-862.