Implementation of GPS Based Trajectory Control of an Autonomous Sailboat

by

Jackson O. Wirekoh

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN MECHANICAL ENGINEERING
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2013

Signature of Author: _____
Department of Mechanical Engineering
May 10, 2013

Certified by: _____
John J. Leonard
Professor of Mechanical and Ocean Engineering
Thesis Supervisor

Accepted by: _____
Annette Hosoi
Professor of Mechanical Engineering
Undergraduate Office

Implementation of GPS Based Trajectory Control of an Autonomous Sailboat

by

Jackson O. Wirekoh

Submitted to the Department of Mechanical Engineering
on May 10, 2013 in Partial Fulfillment of
the Requirements for the Degree of Bachelor of Science in
Mechanical Engineering

Abstract

Autonomous robotic systems are increasingly becoming a major component of modern society. In order to gain a better understanding of the capabilities of these autonomous systems, experimentation was conducted using a miniature robotic sailboat. GPS based trajectory control was implemented on this system to test the ability of the sailboat to travel to desired locations when placed in different starting positions. Ultimately, the sailboat was able to travel the desired 30 feet in the shortest amount of time when it began with its angle heading projected towards the desired location without any obstacles to avoid.

Thesis Supervisor: John J. Leonard
Title: Professor of Mechanical and Ocean Engineering

Table of Contents

Abstract 2

1. Introduction 4

2. Background 4-5

    2.1. Sailing 4-5

    2.2. GPS 5

3. Experimental Apparatus 6

    3.1. Mechanical Design 6-7

        3.1.1. System Architecture 6

        3.1.2. Mechanical Power 7

    3.2. Electrical Design 7-8

        3.2.1. Circuit 7

        3.2.2. Electrical Power 7

        3.2.3. Electrical Loads 7-8

    3.3 Software Design 8

4. Experimental Procedure 8

5. Results & Conclusion 8-9

Bibliography 10

Appendix A 11-15

Appendix B 16-17

Appendix C 18-24

1. Introduction

Robotic systems have become an integral part of modern society. The numerous capabilities and versatility of said systems have made them a cornerstone in industry, as well as in the home. These systems aid humans across the planet with manufacturing, surveillance, home maintenance, as well as in other various sectors of life. In many cases, these systems complete their tasks by following predetermined parameters. This includes preprogrammed paths of motion that the robot will enact continuously until its power supply has been depleted or it has been directed to stop.

In other cases these systems act autonomously. Autonomous robotic systems are capable of reacting to their environments in ways that the systems previously mentioned are not. These autonomous systems are capable of analyzing their environment using sensors, and then executing an action that best fits their needs. This allows for robotic systems to perform complex tasks, such as mowing a lawn, deep sea exploration, or autonomous navigation, the latter being explored throughout this study. Using a handmade sailboat as the plant, Global Positioning Satellite based control was enacted to provide the sailboat with coherent autonomous logic. Furthermore, the ability of the sailboat to traverse obstacles en route to desired locations was tested.

2. Background

2.1. Sailing

Sailing is a form of propulsion which is powered by the force and speed of the wind. This method of propulsion has been employed since the 6th millennium B.C., and has been instrumental in allowing humanity to expand its earlier civilizations around the world. The impact of sailing has been most profound in maritime travel. With the creation of vessels capable of using the wind for both power and control, seamen gained a valuable means to navigate the Earth's seas. These vessels, which are today known as sailboats, combined various systems to make wind-aided travel both safe and reliable. Figure 1 provides a diagram of a sailboat.

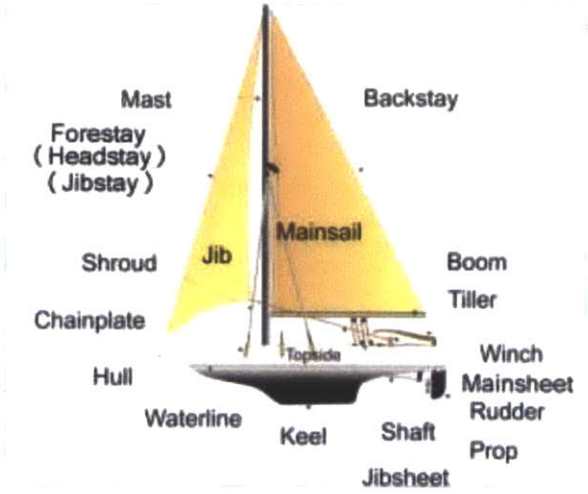


Figure 1: Diagram of the structure of a simple sailboat.[1]

A typical sailboat is composed of four major components. These components include the hull, the keel, the masts/sails, and the rudder. The hull, which can be produced from a mixture of wood, metal, fiberglass, or composite materials, is the frame of the sailboat. It provides the boat with its unique shape and is designed to ensure the boat can remain buoyant through tumultuous sea conditions. Sailboats can be designed with numerous different hull designs. The most common, which can be seen in Figure 2, is known as a monohull. In addition, Figure 3 provides an example of a multihull vessel.

Figure 2: Image of typical monohull design, which is incredibly resistant to capsizing.[2]

Figure 3: Image of a trimaran, one of many different types of multihull designs.[3]

The keel forms the backbone of the hull, as well as a hydrodynamic balancing agent for the boat. A weight is fitted to the endpoint of the keel, to serve as ballast in order to counteract the leeway force of the wind on the sails. Subsequently, the sails provide the major power source for the sailboat. Utilizing various configurations and sizes, the sails can be used to change the direction that the boat travels in, as well as the speed of the boat relative to the wind force. Additionally, skilled seamen are capable of traversing in the opposite direction of the wind (upwind) by tacking. Alongside the rudder, which allows a boat to turn around its center of mass, the sails can be maneuvered to propel the boat upwind at varying angles of attack. This versatility, when coupled with proper navigation systems has produced an innovative maritime creation.

2.2. Global Positioning Satellite

Global Positioning Satellite, or GPS, is a satellite based navigation system that was originally developed for the United States Department of Defense in 1973, and has since found its way into multiple facets of modern society. Currently GPS operates using 24 to 32 Intermediate Circular Orbit (ICO) satellites, at altitudes between 2000 kilometers and 35,786 kilometers, to perform measurements. Through the use of these satellites revolving around the Earth, the latitude, longitude, and altitude of any point on the planet can be pinpointed. Figure 4 provides a representation of the orbital satellite system used.

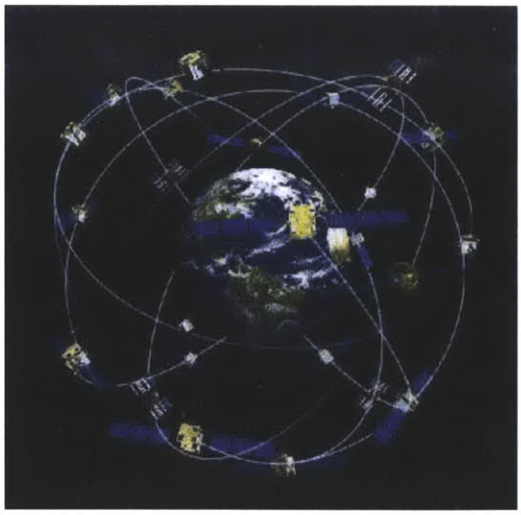


Figure 4: Artistic representation of the ICO satellite system used to calculate GPS based location on Earth.[4]

In order for these measurements to be accurate however, at least four satellites must be detected by the user operated GPS receiver (i.e. cell phone, car GPS, etc.). Once the appropriate number of satellites has been detected, a GPS receiver requests information from the satellites including the time of the message and the location of the satellite when the message was sent. This allows the GPS unit to calculate the transit time of each message and calculate the distance to each satellite using the speed of light c, and the navigation equation

$$\left((\tilde{t}_r + b - t_i)c\right)^2 = (x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 \quad (1)$$

where x_i, y_i, and z_i, are components of the satellite position, t_i is the time component of the satellite, b is the receiver's time delay, $\tilde{t}_r$ is the true reception time, and i ranges from 1 to at least 4 different satellites. Equation 1 can then be solved for the receiver position x, y, and z, and$b$.

GPS based navigation provides accurate high resolution coordinates, in addition to being effective in all weather conditions. As long the line of sight between the GPS receiver and at least four satellites is clear, the GPS will produce reliable information for navigational purposes. These factors have made GPS based control ideal for this study.

3. Experimental Apparatus

GPS based trajectory control was implemented on a model sailboat, the experimental apparatus, in order to allow the sailboat to travel to desired locations autonomously. The system was outfitted with sensors, motors, and servos in order to provide enough freedom for the boat to operate efficiently. Subsequent sections detail the mechanical, electrical, and software design components of the experimental apparatus.

3.1. Mechanical Design

3.1.1. System Architecture

The design of the sailboat was inspired by the MIT Tech racing sailboat. The hull was directly modeled from the hull lines used to create the Tech sailboat with minor adjustments. The hull is 18" x 8" x 3" at its longest points, and was created from three 3D printed sections made of ABS plastic. These sections were connected using a combination of epoxy, fiberglass, and bolts. In addition, casings and inserts were designed into the hull to hold the mast, keel, rudder, and electronics (Arduino, servos, motors, sensors, etc.). Figure 5 provides a solid model of the hull design.

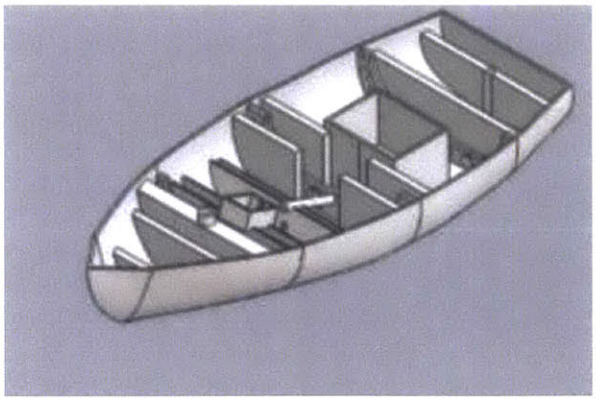


Figure 5: SolidWorks model of the monohull used. Ribs were designed to provide stability to the hull, while different sections were created to various components.

Other main features of the boat are modified versions of the keel, rudder, and mast/sail system incorporated into the Tech design. Figure 6 provides a dimensioned solid model of the keel, while Figure 7 provides a scaled image of the actual boat. The keel is composed of birch wood connected to an iron mass by epoxy and fiberglass, while the rudder is composed of a balsa wood rod connected to a birch wood baseplate. The mast is made from balsa wood and is connected to the Mylar sails by sailing tape. Furthermore, these sails can be adjusted to optimize the wind force on the boat.

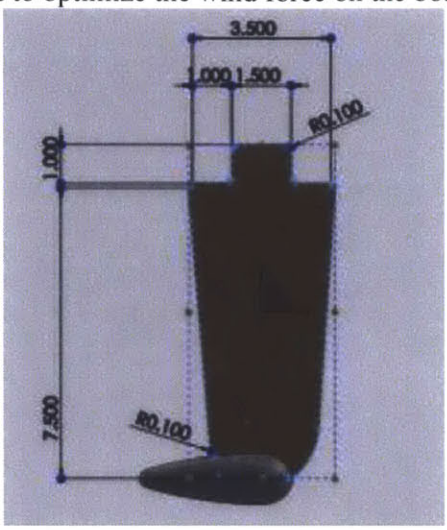


Figure 6: Dimensioned solid model of the Keel which is formed by joining a wooden base plate to a weight through a combination of a rabbet joint and fiberglass.

Figure 7: Scaled image of the completed Sailboat.

3.1.2. Mechanical Power

Power is provided to the sailboat through two means. First, the sails drive the sailboat through the use of wind power. Adjusting the sails allows the boat to flow with the wind when traveling downwind or tack while traveling upwind. A secondary power source is provided by a brushed DC motor capable of producing a speed of 1 knot. The DC motor is connected by a jack shaft to a plastic propeller in order to produce enough force to move the sailboat forward.

3.2 Electrical Design

3.2.1. Circuit

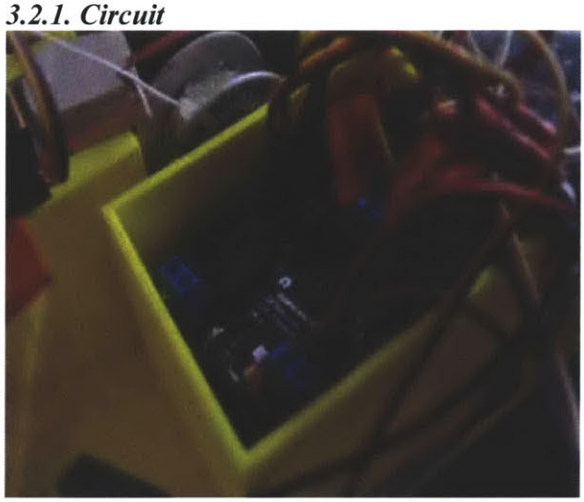


Figure 8: Image of a portion the integrated circuit used to operate the sailboat.

The circuitry for this system employs a complex parallel circuit. A battery pack, providing 7.4 volts, powers an Arduino Uno microcontroller board, an XBee electronics shield, and an Adafruit GPS shield. Subsequently these boards provide power to 3 servo motors, a brushed DC motor, 3 IR Sensors, and 2 anemometers. Graphical representation of the circuitry elements can be found in Appendix B.

3.2.2. Electrical Power

Electrical power is provided to the circuit by a rechargeable Lithium Polymer battery pack. The battery pack generates 7.4 volts for the system and holds an expected life of 1900 milliamp-hours. Figure 9 provides an image of the battery used.

Figure 9: Thunder Power DC LIPO 1900mAh battery.[5]

Based on these specifications, this battery provides a reasonable power supply for the boat. Under operating conditions, the sailboat can be powered for up to 2 hours.

3.2.3. Electrical Loads

Three HS-322HD Standard Deluxe servos, depicted in Figure 10, are used to control different onboard systems. These servos operate the rudder, turning the boat in desired directions, the wench, which pulls the sails in and out, and rotate the pseudo-LIDAR system at the front of the boat. This servo was chosen to complete these tasks due to its 180° range of motion as well as its ability to provide enough torque to compete with forces felt on the sails or rudder. Additionally, these servos operate between 4.6V and 6V which is in the range of the battery.

Figure 10: A HS-322HD Standard Deluxe motor, similar to the servos used to operate the LIDAR, wench, and rudder of the sailboat.[6]

A 110 RPPM Micro Gearmotor is also integrated into the system. This actuator has a stall torque of 32 oz-in and can be digitally controlled by the Arduino Uno. Running at an operational voltage of 6V, this motor is capable of producing a velocity of up to 1 knot. Other

loads include the 3 Sharp IR Sensors and 2 Modern Devices anemometers. The IR Sensors are medium range distance sensors that can accurately measure distances between 4 inches and 34 inches. The anemometers measure the speed of incoming wind, producing larger voltage readings with increasing wind velocity. These devices both operate at 5V, and provide readable electrical signals to the Arduino microcontroller. Further specifications, in addition to images, for the HS-322HD Servo, the Micro Gearmotor, and the anemometer can be found in Appendix A.

3.3 Software Design

Using Arduino Processing language, a program was created to provide the sailboat with autonomous navigational operation. Initially this program waits until enough satellites have been detected for the Adafruit GPS shield. Once this has occurred, the system sets the current location of the sailboat as the home position, and begins calculating its trajectory to the desired location. However, should there be any form of impedance detected by the IR sensors; the sailboat will enact an obstacle avoidance program before plotting a new trajectory to the desired location. This trajectory is continually updated using intermediate GPS measurements of the sailboats current location, in order to ensure that actuators work together to provide the sailboat the optimal trajectory. Figure 11 provides a graphical representation of the values used in the control logic.

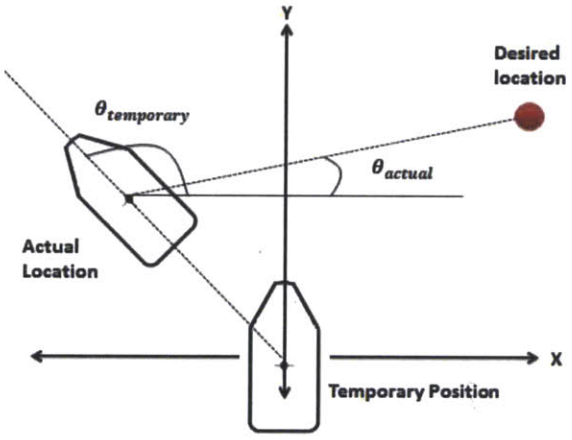


Figure 11: Displays a graphical representation of control logic. Sailboat determines required angle heading change by finding the angle difference between the temporary position and actual position, as well as the actual position and the desired position.

This control logic was chosen due to the difficulty of controlling this system without a sensor to detect the angle heading of the boat. By using a temporary position, calculated at about 10 second intervals, and the equations,

$$\theta_{temp} = \tan^{-1}\left(\frac{y_{act} - y_{temp}}{x_{act} - x_{temp}}\right) \quad (2)$$

$$\theta_{act} = \tan^{-1}\left(\frac{y_{desired} - y_{act}}{x_{desired} - x_{act}}\right) \quad (3)$$

the controller detects which direction the boat is moving in. Additionally, the desired angle heading is also calculated. This ensures that the sailboat understands how long it must have both the motor on and the rudder turned in order to reach the desired heading. Further information on the control logic and implementation can be found in Appendix B, while the actual program is found in Appendix C.

4. Experimental Procedure

The ability of this controller to travel to a desired location 30 feet away was tested. In addition, the time it took to reach the destination, was recorded and compared to a theoretical value equal to 17.775 seconds. This value was calculated based on the speed of the sail boat (1 knot) and the total distance it had to travel (30 feet). A total of 30 trials were conducted. 5 trials were conducted with the heading of the boat pointed in the desired direction, with the heading offset by 45 degrees, and with the heading offset by -45 degrees. In addition, these trails were also conducted with an obstacle blocking the path of the sailboat at 10 feet away.

5. Results & Conclusion

The time it took to travel to the desired location was the least, when the boat heading was aligned with the desired point. It took the boat an average 20.12 seconds to reach the desired location. Further results can be viewed in Table1 and Table 2.

Table 1: Data Collected

Trial Start Position	Test 1	Test 2	Test 3	Test 4	Test 5
-45° offset	20.9058	20.9134	21.6324	21.0975	21.785
Head on	20.5469	19.9575	19.9649	20.1576	19.9706
45° offset	20.9572	21.4854	20.8003	21.1419	21.4218
-45° with obstacle	24.8147	24.9058	25.1270	24.9134	25.6324
Head on with obstacle	23.0975	23.2785	23.5469	22.9575	22.9649
45° with obstacle	25.1576	24.9706	24.9572	24.4854	24.8003

Table 2: Average

Start Position	Average (s)
-45° offset	21.267 ± .414
Head on	20.12 ± .253
45° offset	21.161 ± .294
-45° with obstacle	25.079 ± .33
Head on with obstacle	23.169 ± .248
45° with obstacle	24.874 ± .252

The sailboat was able to correctly sail to desired locations. However, these locations had to be specified in the program initially. In order to better test the navigational capabilities of the control logic, a system should be installed to allow the user to change desired locations while in the midst of travel.

---

[1] Image taken from: http://www.frugal-mariner.com/Boat_and_Ship_Rigs.html

[2] Image taken from: http://www.sunsail.com/fleet/sunsail-32i-2-cabin-monohull-yacht

[3] Image taken from: http://yachtpals.com/monster-multihulls-9004

[4] Image taken from: http://www.nist.gov/pml/div688/grp40/gpsarchive.cfm

[5] Image taken from: http://www.amazon.com/Thunder-Power-RC-1900mAh-Receiver/dp/B0051BTRC0

[6] Image taken from: http://www.servocity.com/index.html

Bibliography

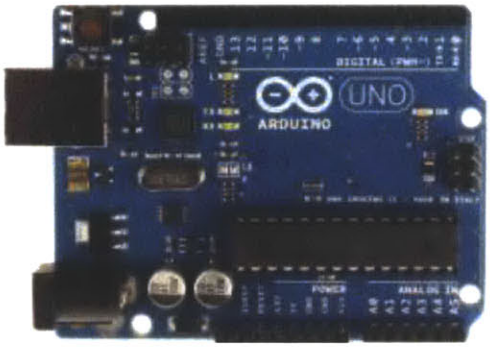
Bancroft, S. (1985). An algebraic Solution of the GPS Equations. *Aerospace and Electronic Systems, IEEE Transactions on, AES-21 (1), 56-59.*http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4104017

Carter, R. (2006). Boat remains and maritime trade in the Persian Gulf during the sixth and fifth millennia BC. *Antiquity, 80 (307), 52-63.http://search.proquest.com.libproxy.mit.edu/docview/217579905?accountid=12492*

Parkinson, B. W. (1996). *Global Positioning System: Theory and Applications, Volume I.* Washington, DC: American Institute of Aeronautics and Astronautics.

Appendix A

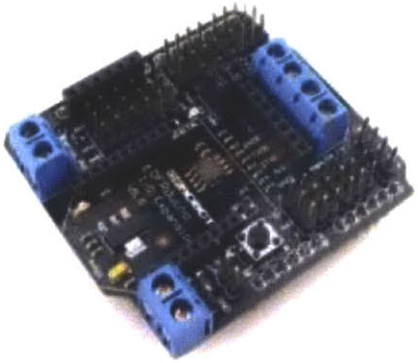
Arduino Uno



Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Site: arduino.cc/en/Main/arduinoBoardUno

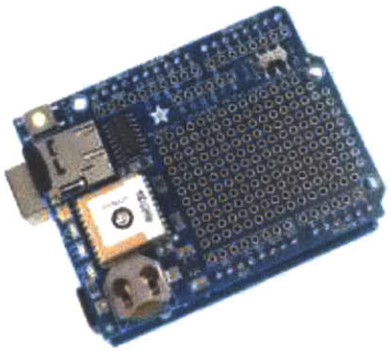
IO Expansion Shield V5 XBee Sensor Shield RS485



- Dimensions: 2.24 in x 2.09 in x 0.71 in (5.7 cm x 5.3 cm x 1.8 cm)
- Weight: 0.92 oz (26 g)
- Extends of 14 digital IO port (12 Servo Interface),power, and 6 analog IO ports and power

Site: http://www.dhgate.com/arduino-xbee-shield-v5-expansion-shield-v5/p-ff8080813916f097013922f589d00646.html

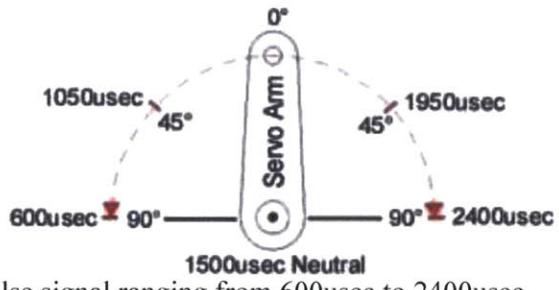
Adafruit GPS Shield



- 165 dBm sensitivity, 10 Hz updates, 66 channels
- Low power module - only 20mA current draw, half of most GPS's
- Assembled & tested shield for Arduino Uno/Duemilanove/Diecimila/Leonardo (not for use with Mega/ADK/Due)
- MicroSD card slot for datalogging onto a removable card
- RTC battery included, for up to 7 years backup
- Built-in datalogging to flashPPS output on fix>25Km altitude
- Internal patch antenna + u.FL connector for external active antenna
- Power, Pin #13 and Fix status LED
- Big prototyping area
- Dimensions(PCB only): 69mm x 53mm x 6.7mm (2.7in x 2.1in x 0.26in)
- Weight (w/o GPS module): 24g

Site: http://www.adafruit.com/products/1272#Technical Details

HS-322HD Standard Deluxe

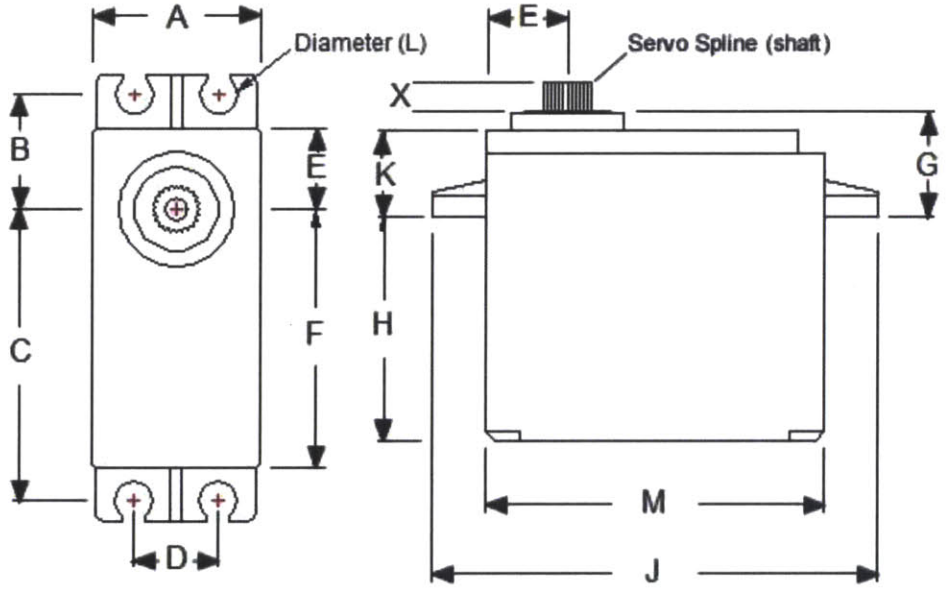


This servo can operate 180° when given a pulse signal ranging from 600usec to 2400usec.

Detailed Specifications

Control System: +Pulse Width Control 1500usec Neutral
Required Pulse: 3-5 Volt Peak to Peak Square Wave
Operating Voltage: 4.8-6.0 Volts

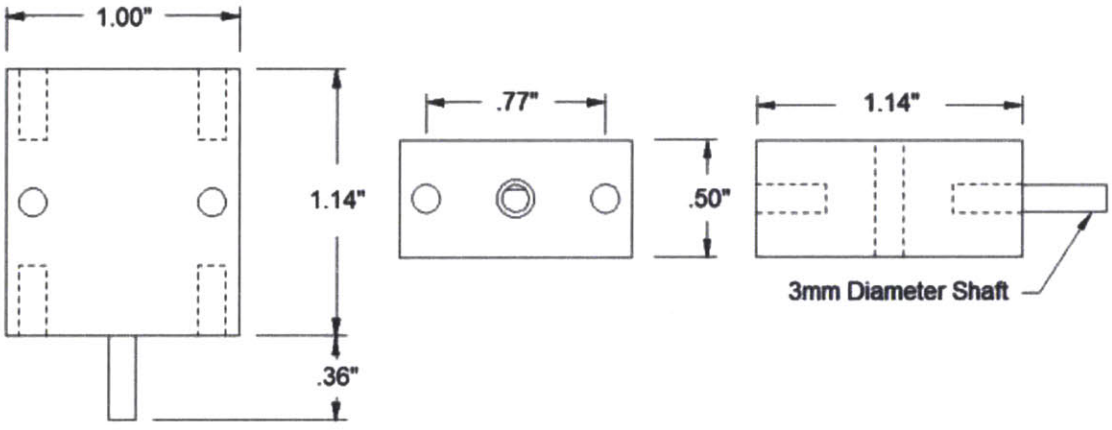
Operating Temperature Range: -20 to +60 Degree C
Operating Speed (4.8V): 0.19sec/60° at no load
Operating Speed (6.0V): 0.15sec/60° at no load
Stall Torque (4.8V): 42 oz/in (3.0 kg/cm)
Stall Torque (6.0V): 51 oz/in (3.7 kg/cm)
Current Drain (4.8V): 7.4mA/idle and 160mA no load operating
Current Drain (6.0V): 7.7mA/idle and 180mA no load operating
Dead Band Width: 5usec
Operating Angle: 40 Deg. one side pulse traveling 400usec
Direction: Clockwise/Pulse Traveling 1500 to 1900usec
Motor Type: Cored Metal Brush
Potentiometer Drive: 4 Slider/Direct Drive
Bearing Type: Top/Resin Bushing
Gear Type: Heavy Duty Resin
360 Modifiable: Yes
Connector Wire Length: 11.81" (300mm)
Dimensions: See Schematics
Weight: 1.52oz (43g)
Site: http://www.servocity.com/index.html



A = .780" (19.82mm)
B = .530" (13.47mm)
C = 1.33" (33.79mm)
D = .400" (10.17mm)
E = .380" (9.66mm)
F = 1.19" (30.22mm)
G = .460" (11.68mm)
H = 1.05" (26.67mm)
J = 2.08" (52.84mm)
K = .368" (9.35mm)
L = .172" (4.38mm)
M = 1.57" (39.88mm)
X = .120" (3.05mm)

110 RPM Micro GearmotorBlocks

Operating Range: 6VDC
Torque @ Stall: 32 oz-in.@ 6VDC
Torque @ Stall: 60 oz-in.@ 12VDC
0.118" (3mm) Diameter "D" Shaft
No load current: 30mA (6VDC)
No load current: 70mA (12VDC)
Stall current: 360mA (6VDC)
Stall current: 1600mA (12VDC)
No load speed: **55 RPM @ 6VDC**
No load speed: **110 RPM @ 12VDC**
Gear ratio: 250:1
Motor size: 1.14" x 1.00" x 0.50"
Shaft size: 0.118" (3mm) Dia. x 0.393" (10mm) L
Weight: 0.6 oz (17g)
DC reversible motors
Pre-tinned wire leads (6")
Site: http://www.servocity.com/html/110_rpm_micro_gearmotorblocks.html



Sharp IR Sensor 2Y0A21F1Y

The Sharp GP2Y0A21YK0F infrared distance measuring sensor uses a beam of infrared light to reflect off an object to measure its distance. Because it uses triangulation of the beam of light to calculate the distance, it is able to provide consistent and reliable readings which are less sensitive to temperature variation or the object's reflectivity. The sensor outputs an analog voltage corresponding to the distance of the object, and can easily be read using an inexpensive analog to digital converter (ADC) chip.
The Sharp IR Sensor Stand kit (725-28995) provides a very convenient method of mounting the sensor, and the Sharp IR Sensor to Servo Cable (805-28995) makes it easy to wire up to a breadboard or microcontroller board.

Features:

- Distance measurement range: 10 to 80 cm (3.9 to 31.5 inches)
- Analog output voltage corresponds to distance
- Operates on 5 V supply
- Convenient 3-pin interface
- Two mounting holes spaced 1.46 inches (37 mm) apart

Application Ideas:

- Distance sensor for autonomous robots
- Non-contact optical switch
- Industrial automation and controls

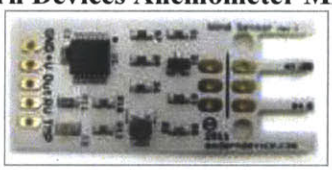
Key Specifications:

Power requirements: 5 VDC, 30 mA (typ.)

- Communication: Analog output voltage (typically 0.4 to 2.3 V range)
- Overall Dimensions: 1.75 x 0.74 x 0.53 in (4.45 x 1.89 x 1.35 cm)
- Operating temp range: +14 to +140 °F (-10 to +60 °C)

Site:
http://www.parallax.com/StoreSearchResults/tabid/768/List/0/SortField/4/ProductID/776/Default.aspx?txtSearch=sharp+ir+sensor

Modern Devices Anemometer MD0550

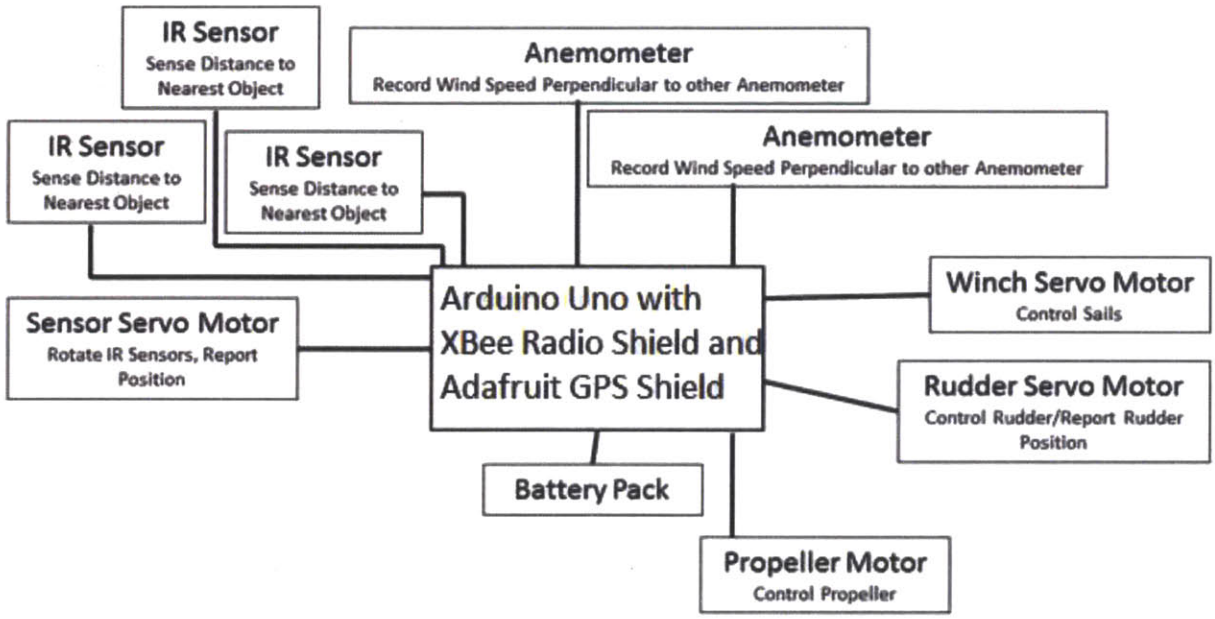


Specifications

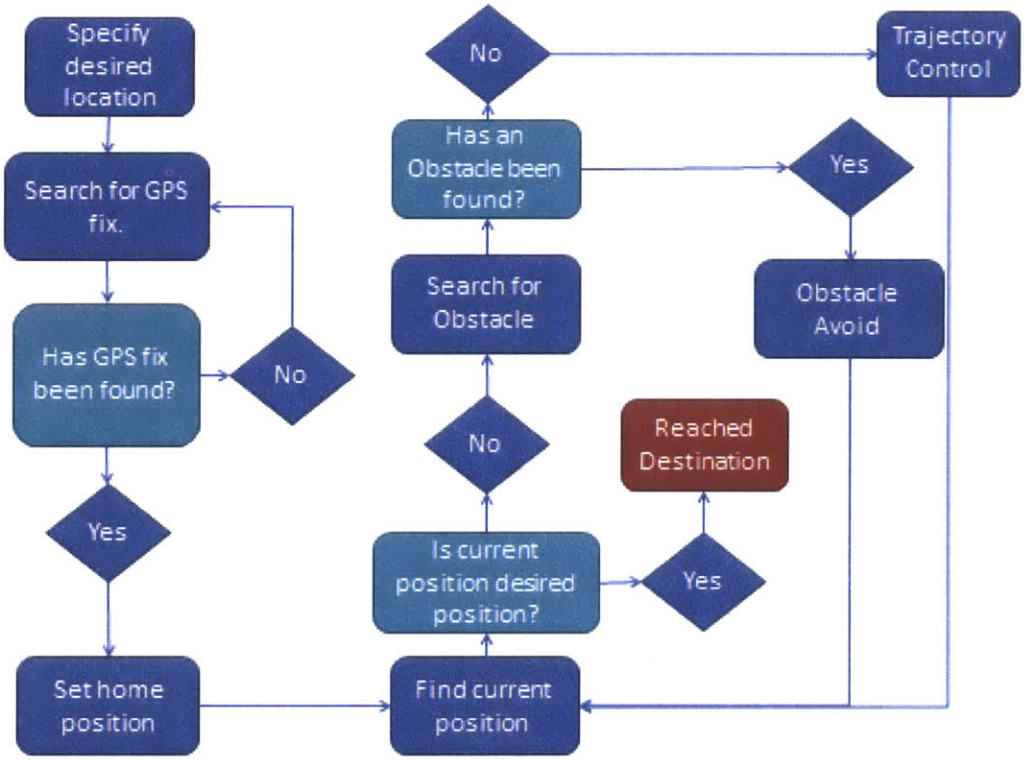
- Dimensions: .68″ × 1.590″ × .25″
- Supply Voltage: 4 – 10 volts
- Supply current: 20 – 40 mA (depending on wind speed)
- Output signal: analog, 0 to VCC

Site: http://shop.moderndevice.com/products/wind-sensor

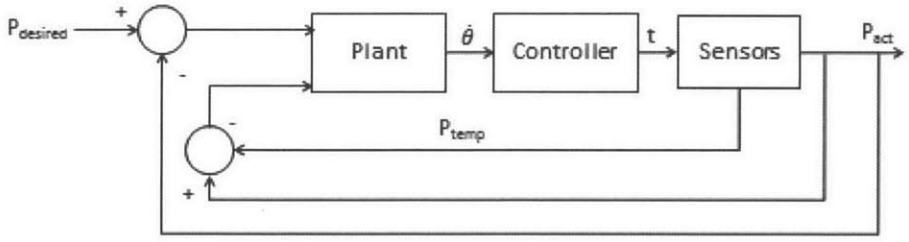
Appendix B
Circuit Diagram



Logic Diagram



Control Diagram



Appendix C

```
/**
* @file: SailboatCode
 * @author: Jackson Wirekoh (jwirekoh@mit.edu)
 * @date: 3/27/2013
 *
 * @description: Controls components of autnomous Sailboat
 *
 * @hardware: 3 Servos, 1 Dc motor, 1 GPS chip, 1 gyroscope, 3 IR Sensors, 2 Wind Sensors
 * @software: Processing
 * @libraries: Servo, Software, Adafruit_GPS, Math
**/

#include <Servo.h>
#include <SoftwareSerial.h>
#include <Adafruit_GPS.h>
#include <math.h>



// Creation and implementation of servo, sensor, and actuator components

// Create Servo Objects
Servo lidarServo;
Servo sailServo;
Servo rudderServo;
// Assign Output Pins
int ls=2;
int ss=3;
int motorOut=4;
int rs=5;

// Set position values for Servos
int neutral = 1485;
int left = 1000;
int right = 1800;
int in =2400;
int out = 1000;

// Assign Input Pins
int motorIn=A5;
int centerIR=A4;
int leftIR=A3;
int rightIR=A2;
int frontWind=A1;
int backWind=A0;

// Declare variables to read inputs
int cenRead=0, leftRead=0, rightRead=0, fWind=0, bWind=0;

// Creation and implementation of GPS and SoftwareSerial objects
```

```
// Create GPS Objects
SoftwareSerial mySerial(8,7);
Adafruit_GPS GPS(&mySerial);

// Set GPSECHO to 'false' to turn off echoing the GPS data to the Serial console
#define GPSECHO  false

// InComplete
  uint32_t timer = millis();
// Setup boolean value
boolean findPosition = true;

//Set home position
float xh,yh;
// Set desired locations
float xd=7105.07042, yd=4220.63238;
//Set actual values
float x,y;
float theta;
float actDist;
// trajectory control values
float xtemp, ytemp, thetaTemp, tempDist;
// Boat speed during motor on in meters pper second
float velocity = .5144;  // meters/sec
float radius = .2286; // distance in meters around which boat spins
float angVel = velocity/radius*180; // in degrees per second


// Setup Pins
void setup(){
   Serial.begin(115200);     //Send data at 9600 bps(bits per second)

  //Sets pins as outputs
  //Pins will not take input from soure (i.e. computer)
  pinMode(ls,OUTPUT);
  pinMode(ss,OUTPUT);
  pinMode(motorOut,OUTPUT);
  pinMode(rs,OUTPUT);

  // Attach servos to ouput pins
  lidarServo.attach(ls);
  sailServo.attach(ss);
  rudderServo.attach(rs);
  // Set Servos to neutral postition
  lidarServo.writeMicroseconds(1500);
  sailServo.writeMicroseconds(neutral);
  rudderServo.writeMicroseconds(neutral);
  Serial.println("Servos Ready");
  delay(800);
```

```
  //Sets pins as inputs
  // pins will take readings from sensors
  pinMode(motorIn,INPUT);
  pinMode(centerIR,INPUT);
  pinMode(leftIR,INPUT);
  pinMode(rightIR,INPUT);
  pinMode(frontWind,INPUT);
  pinMode(backWind,INPUT);
  Serial.println("Sensors Ready");
  delay(800);

  // 9600 NMEA is the default baud rate for Adafruit MTK GPS's- some use 4800
  GPS.begin(9600);
  // uncomment this line to turn on RMC (recommended minimum) and GGA (fix data) including altitude
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
  // Set the update rate
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);   // 1 Hz update rate
  // For the parsing code to work nicely and have time to sort thru the data, and
  // Request updates on antenna status, comment out to keep quiet
  GPS.sendCommand(PGCMD_ANTENNA);

  // the nice thing about this code is you can have a timer0 interrupt go off
  // every 1 millisecond, and read data from the GPS for you. that makes the
  // loop code a heck of a lot easier!

  delay(800);
  // Ask for firmware version
  mySerial.println(PMTK_Q_RELEASE);
  delay(800);

  while(findPosition){position();}
  findPosition=true;

  xh=GPS.longitude; yh=GPS.latitude;
  Serial.print("Home: ");
  Serial.print(xh,4); Serial.print(", ");
  Serial.println(yh,4);
}

int counter=0;
void loop(){
    digitalWrite(motorOut,HIGH);
  //find its current position
  while(findPosition){position();}
  if (counter%2==0){
     xtemp=GPS.longitude;
     ytemp=GPS.latitude;
  }
  // If current position is equal to desired position stop and come back home
   if(abs(xd-x)<=.0001 && abs(yd-y)<=.0001){
    delay(2000);
```

```
    xd=xh;
    yd=yh;
    if(abs(xh-x)<=.0001 && abs(yh-y)<=.0001){
      Serial.println("I'm Home");
    }
  }
  // If boat is not at destination do this
  else{
    // Check to see if there are any obstacles in the way
    senseObstacle();
    //if there are obstaacles begin obstacle avoidance
    if (cenRead<10 ||leftRead<10 || rightRead<10){
      obstacleAvoid;
    }
    // If no obstacles calculate and follow trajectory to desired location
    else{
    travel();
    }
    findPosition=true;
    counter++;
  }
}

// Test end


// May not be used
void upWind(){
  sailServo.writeMicroseconds(in);
  delay(200);
  digitalWrite(motorOut,HIGH);
  delay(200);
  //Tack
}

//May not be used
void downWind(){
  digitalWrite(motorOut,LOW);
  delay(200);
  sailServo.writeMicroseconds(out);
  delay(200);
}


// command used to avoid obstacles
void obstacleAvoid(){
  sailServo.writeMicroseconds(in);                    //Pull in Sails to begin pbstacle avoidance
  delay(200);

  // Decide on direction to turn
  if (cenRead<12 && rightRead<12 || rightRead<12){
```

```
   rudderServo.writeMicroseconds(left);
  }
  else if(cenRead<12 && leftRead<12 || leftRead<12){
   rudderServo.writeMicroseconds(right);
  }
  //else if(cenRead<12 &&leftRead>12&& rightRead>12){}
  else{ delay(500);}

 // Begin avoiding obstacles using motor for speed
  digitalWrite(motorOut,HIGH);
  delay(200);

}

// command used to request information from IR Sensors
void senseObstacle(){
  Serial.println("Sensing Obstacles");
  // Read Analog inputs
  cenRead=analogRead(centerIR);
  leftRead=analogRead(leftIR);
  rightRead=analogRead(rightIR);
  delay(200);
  // Map analog values to inches
  cenRead=map(cenRead,0,600,31,6);
  leftRead=map(leftRead,0,600,31,6);
  rightRead=map(rightRead,0,600,31,6);
  delay(500);
  Serial.println("left: ");
  Serial.println(leftRead);
  Serial.println("center: ");
  Serial.println(cenRead);
   Serial.println("right: ");
  Serial.println(rightRead);
  delay(1000);
}

void position(){
  // read data from the GPS in the 'main loop'
    char c = GPS.read();
    // if you want to debug, this is a good time to do it!
    if (GPSECHO)
      if (c) Serial.print(c);
  if (GPS.newNMEAreceived()) {
    // a tricky thing here is if we print the NMEA sentence, or data
    // we end up not listening and catching other sentences!
    // so be very wary if using OUTPUT_ALLDATA and trytng to print out data
    //Serial.println(GPS.lastNMEA());   // this also sets the newNMEAreceived() flag to false

    if (!GPS.parse(GPS.lastNMEA()))   // this also sets the newNMEAreceived() flag to false
      return;  // we can fail to parse a sentence in which case we should just wait for another
  }
```

```
  // if millis() or timer wraps around, we'll just reset it
  if (timer > millis())  timer = millis();

  // approximately every 2 seconds or so, print out the current stats
  if (millis() - timer > 5000) {
    timer = millis(); // reset the timer
    if (GPS.fix) {
      x=GPS.longitude; y=GPS.latitude;
      Serial.print("Location: ");
      Serial.print(GPS.longitude, 4); Serial.print(GPS.lon);
      Serial.print(", ");
      Serial.print(GPS.latitude, 4);Serial.println(GPS.lat);
     // Serial.println(GPS.angle);

      findPosition=false;
    }
  }
}

float calcAngle(float x1, float y1, float x2, float y2){
  float ydist=(int(y2/100)+(y2/100-int(y2/100))/.600)-(int(y1/100)+(y1/100-int(y1/100))/.600);
  float xdist=(int(x2/100)+(x2/100-int(x2/100))/.600)-(int(x1/100)+(x1/100-int(x1/100))/.600);
  float calcTheta;
  if(ydist>=0 && xdist>=0){calcTheta=atan(ydist/xdist)*180/M_PI;}
  else if(ydist>=0 && xdist<0){calcTheta=180+atan(ydist/xdist)*180/M_PI;}
  else if(ydist<0 && xdist<0){calcTheta=180+atan(ydist/xdist)*180/M_PI;}
  else if(ydist<0 && xdist>=0){calcTheta=360+atan(ydist/xdist)*180/M_PI;}
  return calcTheta;
}
// determine travel trajectory
void travel(){
  thetaTemp=calcAngle(xtemp,ytemp,x,y);
  theta=calcAngle(x,y,xd,yd);
  float thetaDiff=(theta-thetaTemp);
  float wait=0;

  if (thetaTemp>theta && abs(thetaDiff)<=180){
    rudderServo.writeMicroseconds(right);
    wait=abs(theta-thetaTemp)/angVel*1000;
  }
  else if(thetaTemp>theta && abs(thetaDiff)>180){
    rudderServo.writeMicroseconds(left);
    wait=(360-abs(theta-thetaTemp))/angVel*1000;
  }
  if (thetaTemp<theta && abs(thetaDiff)<=180){
    rudderServo.writeMicroseconds(left);
    wait=abs(theta-thetaTemp)/angVel*1000;
  }
  else if(thetaTemp<theta && abs(thetaDiff)>180){
    rudderServo.writeMicroseconds(left);
```

```
    wait=(360-abs(theta-thetaTemp))/angVel*1000;
  }
  Serial.print("wait: "); Serial.println(int(wait));
  Serial.print("angular velocity: "); Serial.println(angVel);
  digitalWrite(motorOut,HIGH);
  delay(wait);
  rudderServo.writeMicroseconds(neutral);
}
```