Vectorization of GPS mapping system using given coordinates

Chris Clark

ECE 478

Homework 2

Problem Statement: How to vectorize any given two sets of GPS coordinates for best possible trajectory outcome.

 As the problem states we are looking for a simple method for vector analysis given coordinates within the system. The way that we map and divide the world up, given the latitude and longitude of any given object sets up the perfect coordinate grid.

 If we set our current position as the origin of the coordinate grid we can effectively set up a simple vector that can direct an object to the goal with the shortest possible path.

 Pseudo Code for analysis:

 Float lat\_goal; //the latitude of our goal

 Float long\_goal; // the longitude of our goal

 Float lat\_current; // the latitude of our current position

 Float long\_current; // the longitude of our current position

 Float error\_lat; // the error that we are assuming for latitudinal positioning

 Float error\_long; // the error that we are assuming for longitudinal positioning

 Gps\_data[] = “”

We will get these values based on the input string coming in from the gps sensing unit. It gets read in through the serial port and the data will then be parsed to separate out the latitude and longitude of the position. This will be done continuously while the robot is moving and the current position is not equal to the goal position.

The while loop will look like:

 While ((lat\_goal != (lat\_current + error\_lat)||(lat\_current – error\_lat)) && (long\_goal != ((long\_current +error\_long) || (long\_current – error\_long))))

Within this loop we will continually get the current position of the sensor within the movement to re-evaluate the loop. This also allows for other operations to take place as the robot will not be moving very fast so we can set it on a path and it will continue on the path while we do other incremental sensor processing.

One thing this loop provides us is incremental changes to the directionality of the robot at large. Using the four variables we set up at the top we would first need to do another read from the serial port and then parse that data again for a quick reset of the direction vector. This will keep it so even if there are mechanical issues within the design it is self-correcting.

 Read\_in();

 Parse\_data(lat\_current, long\_current);

 If (lat\_current > lat\_goal)

 Increase power to motor; //I am assuming power

 If (lat\_current < lat\_goal)

 Decrease power;

We would do this for the longitude of the robot as well so it would be a completely self-sustaining system with minimal process checks due to a set sampling rate. Also what this allows is a “speed” function that if we are a certain distance away we can set our PWM or frequency of voltage higher increasing overall robotic speed to its destination. It could use a pyramid model that until it came within, say, 3-5 ft of the destination it traveled at one speed and then within that range it would travel at another speed. This could be done in multiple steps helping our selection criteria along.

With the data coming in on the constant read from the GPS signals we can vectorize the signal using a subtraction of the signals that we need and those that we have set for our goal. This simple subtraction of our two coordinates would give us a vector with x and y directionality within the coordinate system we have set up using native latitude and longitude as axes. The value of that vector would then determine speed. It would be changing every second as we polled our sensor and reset our values which would give our robot ample time for a live video feed, sonar sensors, and range finders as we traverse the competition grounds.

Code for Gps Main Loop to get data:

**GPS.cpp**

int GPS::sample() { //Gets our sample data for the GPS framework

 float time;

 char ns, ew;

 int lock;

 while(1) {

 getline();

 // Check if it is a GPGGA msg (matches both locked and non-locked msg)

 if(sscanf(msg, "GPGGA,%f,%f,%c,%f,%c,%d", &time, &latitude, &ns, &longitude, &ew, &lock) >= 1) {

 if(!lock) { //if there is a satellite lock it wil skip this otherwise it will register 0 values for longitude and latitude.

 longitude = 0.0;

 latitude = 0.0;

 return 0;

 } else {

 if(ns == 'S') { latitude \*= -1.0; }

 if(ew == 'W') { longitude \*= -1.0; }

 float degrees = trunc(latitude / 100.0f);

 float minutes = latitude - (degrees \* 100.0f);

 latitude = degrees + minutes / 60.0f;

 degrees = trunc(longitude / 100.0f \* 0.01f);

 minutes = longitude - (degrees \* 100.0f);

 longitude = degrees + minutes / 60.0f;

 return 1;

 }

 }

 }

}

float GPS::trunc(float v) {

 if(v < 0.0) {

 v\*= -1.0;

 v = floor(v); //our data comes in as string data and we want it as float data so this will truncuate it and then round it down. The call comes from a formula which multiplies the value by 100

 v\*=-1.0;

 } else {

 v = floor(v);

 }

 return v;

}

void GPS::getline() {

 while(\_gps.getc() != '$'); // wait for the start of a line

 for(int i=0; i<256; i++) { //while our buffer is not full

 msg[i] = \_gps.getc(); //reads in values for our buffer.

 if(msg[i] == '\r') {

 msg[i] = 0;

 printf("%s\r\n", msg);

 return;

 }

 }

 error("Overflowed message limit"); //buffer full

}

**GPS.h**

#ifndef MBED\_GPS\_H

#define MBED\_GPS\_H

/\* A GPS interface for reading from a Globalsat EM-406 GPS Module \*/

class GPS {

public:

 GPS(PinName tx, PinName rx); // Create the GPS interface, connected to the specified serial port

 int sample(); // Sample the incoming GPS data, returning whether there is a lock

 float longitude; // The longitude (call sample() to set)

 float latitude; // The latitude (call sample() to set)

private:

 float trunc(float v);

 void getline();

 Serial \_gps;

 char msg[256];

};

#endif