SOFTWARE DOCUMENTATION

ELCANO HIGH-LEVEL SYSTEM REDESIGN

Rathtana Duong
Pooja Ravi

Electrical Engineering Capstone
University of Washington, Bothell
June 2018

Table of Contents

Introduction 3

C3 Documentation 4

Defining the methods in C3 4

Void C3_communication_with_C2() 4
long turning_radius_mm(long speed_mmPs) 4
bool test_approach_intersection(long turn_radius_mm, int n) 4
bool test_leave_intersection(long turn_radius_mm, int n) 4
bool test_past_destination(int n) 4
void find_state(long turn_radius_mm, int n) 5
int get_turn_direction_angle(int n) 5
void setup_C3() 5
void loop_C3() 6

C6 Documentation 6

Defining the methods in C6 6

void setup_GPS() 6
bool AcquireGPS(waypoint &gps_position) 6
void C6_communication_with_C2() 6
long GetHeading(void) 6
bool initial_position() 7
void setup_C6() 7
void loop_C6() 7

C4 Documentation 7

Defining the methods in C4 7

*void ConstructNetwork(junction *Map, int MapPoints)* 8
*void GetGoals(junction *nodes , int Goals)* 8
*long distance(int cur_node, int *k, long cur_east_mm, long cur_north_mm, int* perCent)* 8
*void FindClosestRoad(waypoint *start, waypoint *road)* 8
void test_closestRoad() 8
int BuildPath (int j, waypoint start, waypoint* destination)* 8
*int FindPath(waypoint *start, waypoint *destination)* 8
*int PlanPath (waypoint *start, waypoint *destination)* 8
boolean LoadMap(char fileName)* 9
void SelectMap(waypoint currentLocation, char fileName, char* nearestMap)* 9
void initialize_C4() 9
void test_mission() 9
void test_path() 9
void test_distance() 9
void setup_C4() 9

Libraries Used…… 9
LEDTester……10
Serial_Communication……11
ElcanoSerial…… 11
IODue…… 11
Adafruit_NeoPixel…… 11
FusionData…… 12
Common…… 12
Adafruit_GPS-master……12
Adafruit_LSM303DLHC-master……12
Adafruit_L3GD20_U-master……12
Adafruit_Sensor-master……12
FastCRC-master……12
Scheduling System……12
Test Interface……13

Introduction

Originally C3, C4 and C6 were implemented as three separate systems using three different Arduino Megas. The previous system used a UART type communication but was a lot slower since it used three different processors. The High level system re-design ports the three processors i.e., the C3, C4, and C6 onto a single processor (Arduino Due) which has a clock rate of 84 MHz. This way, the space for the boards is optimized and the processors can run a lot faster. The software implementation for this project included making changes to the existing high level code and designing a scheduling system that gives chance for each of the three processors to run entirely.

C3 Documentation

The C3 system is called the pilot that transmits how fast the vehicle should go and the direction at which the vehicle should turn to the lower level system. Original version of the C3 code assumed the existence of defined curves through which the trike could move, but since C4 currently, only computes the path by connecting straight lines, all the methods in C3 were re-defined.

Methods defined in C3

The methods defined in C3 are as shown below.

void C3_communication_with_C2()

This method sets up the data and serial communication for sending data from C3 to C2. The desired speed and the turn direction is passed into C2 from C3.

long turning_radius_mm(long speed_mmPs)

- This method computes the turning radius of the trike.
- This method takes in the parameter speed_mmPs which is the actual speed at which the trike is moving.
- This method returns twice the minimum turning radius which is a constant defined as 4000 mm.

Note: This calculation is simplified at present but needs to incorporate the actual speed of the trike to calculate the turning radius in future.

bool test_approach_intersection(long turn_radius_mm, int n)

- This method checks whether the trike is entering an intersection from a straight path as a result of which it will need to turn. Whether or not the trike is entering an intersection depends on how much the trike is turning.
- This method takes in two parameters. First, turn_radius_mm which is the turning radius of the vehicle and second, n which refers to the index of the intersection that the trike is entering into.
- This method returns true if the trike is entering into an intersection and false otherwise.

bool test_leave_intersection(long turn_radius _mm, int n)

- This method checks whether the trike is leaving from an intersection from a turn and into a straight path. Whether or not the trike is leaving an intersection depends on how much the trike is turning.
- This method takes in two parameters. First, turn_radius_mm which is the turning radius of the vehicle and second, n which refers to the index of the intersection that the trike is leaving from.

- This method returns true if the trike is leaving from an intersection and false otherwise.

bool test_past_destination(int n)

- This method checks to see if the trike has moved past the destination and missed to take any turn. Destination in this case, is the next intersection that the trike is approaching. In this computation, the shape of the path such as rectangle, skinny rectangle(a rectangle with very small breadth and a much bigger length) or a square is taken into consideration such that if the vehicle is outside a skinny rectangle but is still within the distance to destination then that is not considered as being past the destination.
- This method takes in the parameter n which refers to the index of the intersection that the trike is entering into.
- This method returns true if the trike is past that intersection and false otherwise.

void find_state(long turn_radius_mm, int n)

- This method detects the state of the trike and sets the speed of the trike accordingly. The “State of the trike” refers to the position that the trike is at such as at rest (stop), entering turn and entering an intersection(approach goal).
- This method takes in two parameters. First, turn_radius_mm which is the turning radius of the vehicle and second, n which refers to the index of the intersection that the trike is approaching.

int get_turn_direction_angle(int n)

- This method determines the direction at which the trike is turning in degrees(90 is right and -90 is left) using the state at which the trike is at. Turn angle is computed using the dot product, while turn direction is computed by the cross product.
- This method takes in the parameter n which is the index of the intersection that the trike is approaching.
- This method returns the degrees that the trike is turning at. If the value returned is positive, it means that the trike is turning right. If the value returned is negative, it means the trike is turning left and if the value returned is zero, it means that the trike is moving straight.

void setup_C3()

- The method is the setup that sets the trike’s initial state. The initial state of the trike is set to straight. This method also sets up the first index of the first intersection and then sets up sending data from C3 to C2.

void loop_C3()

- This is the loop for C3 that gets the turning radius of the trike first and then gets the current state of the trike after which it determines the turning direction. Then the C3 system passes the desired speed and the turn direction onto the lower level system (i.e, C2).

Final Notes on C3: Most of the methods written were simplified in order to make it function for the current high level system. C3 has been tested and is able to determine the state of the vehicle, turn direction, inside the turning radius, setting the correct speed for going straight/turning, and reaching the final destination. C3 able to communicate with C2.

C6 Documentation

The C6 system is called the localization that keeps estimating the current position using information from the GPS and dead reckoning. Then the average of the two position is taken using the fuzzy average. This method helps to get a more accurate estimate of the current position.

Methods defined in C6

The methods defined in C6 are as shown below.

void setup_GPS()

- This method holds the setup information for the gps.

bool AcquireGPS(waypoint &gps_position)

- This method gets the actual position from the gps in terms of latitude and longitude.
- This method takes in a waypoint structure which could be the estimated position or the gps_reading. The estimated position is the current position and the gps_reading is the new position that is captured from the gps and is used in dead reckoning.
- This method returns true if the gps captures a valid position and returns false otherwise.

void C6_communication_with_C2()

- This method has the setup information to receive data from the lower level system. C6 receives the actual speed and the wheel angle from the lower level system.

long GetHeading(void)

- This method returns the bearing of the compass in degrees which is used to compute the north and the east vector and updates new position bearing.

bool initial_position()

- This method is used to decide whether a new valid estimate of the current position can be received from the gps. Either the GPGGA or the GPRMC signals are captured by the gps and is used within the code to get an estimate of the current position.
- This method returns true if the gps captured a valid position and false otherwise

void setup_C6()

- This is the setup for the entire C6 system. This method first sets up the gps and then the compass. This method also captures the time that the first position is got along with the old and the new position. This happens only at the beginning. Finally, the data that needs for C6 to receive the actual speed and the wheel angle is setup.

void loop_C6()

- This is the loop for C6 which starts by trying to get a new GPS reading of the current position. Then, dead reckoning is used to get another estimate of the current position. If both the GPS and the dead reckoning gave a successful estimated current position, then the average of the two estimated positions is calculated using the fuzzy cross point. If the GPS was unable to get a new valid estimate of the current position, then only the dead reckoning is used as the new estimated position. The C6 loop keeps track of and updates three positions which include the older position, the old position and the current position of the trike.
- **Note: Since the current version of Elcano Serial does not have the checksum implemented, Serial_Communication.h was written as a work around to it. In addition, it was noticed that using Dead Reckoning alone actually gave a better estimation compared to the GPS and Dead Reckoning in combination with the fuzzy cross point.**

C4 Documentation

The C4 system is called the mapping that picks the closest map to the current estimated position from the GPS, loads it and computes the route to destination.

Methods defined in C4

The methods defined in C4 are as shown below.

void ConstructNetwork(junction *Map, int MapPoints)

- Takes an array of junctions (intersections) and an integer representing the number of junctions in the array as input. Calculates the distance for all of the edges in the junction and updates the distance value from the multiplier value the array is created with to the distance in mm.

void GetGoals(junction *nodes , int Goals)

- Takes the Mission waypoint array and constructs the Goals array for building the path to take. Currently does not use the *Waypoint or Goals arguments, and instead uses the global Mission and Goals arrays. Can be updated to use the arguments passed in or to remove the arguments if needed.

long distance(int cur_node, int *k, long cur_east_mm, long cur_north_mm, int* perCent)

- Takes the Nodes index (i), a pointer to an index (*k), the east and north coordinates in millimeters, and the percentage of the segment completed and calculates the distance. This is used by the FindClosestRoad function.

void FindClosestRoad(waypoint *start, waypoint *road)

- Takes two waypoints, finds the closest junction to the first waypoint and points the second waypoint to that junction.
- For example, if your map has junctions at (1,1), (-1,1), (1, -1), and (-1,-1) and you pass in your starting waypoint as (2,2), the road waypoint will be updated to (1,1), which is the closest junction to (2,2).

void test_closestRoad()

- Test method that prints the mission after calling the FindClosestRoad method.

int BuildPath (int j, waypoint* start, waypoint* destination)
int FindPath(waypoint *start, waypoint *destination)
int PlanPath (waypoint *start, waypoint *destination)

- These three functions work together to implement the A* pathing. They take in start and destination waypoints which are on the road network given in Nodes and set the waypoints on the path starting with Start and ending with Destination.
- A* is traditionally done with pushing and popping node from an Open and Closed list. However, since we have a small number of nodes, we instead reserve a slot on global variables Open and Closed for each node.

boolean LoadMap(char* fileName)

- The LoadMap function loads the map nodes from a file. It takes in the name of the file to load and loads the appropriate map.
- LoadMap returns true if the map was loaded. It returns false if the load failed.

void SelectMap(waypoint currentLocation, char* fileName, char* nearestMap)

- The SelectMap function determines which map to load. It does this by taking in the current location as a waypoint and a string with the name of the file that contains the origins and file names of the maps. The function then determines which origin is closest to the waypoint and returns the filename as a string.
- It assumes that the file is formatted according to the specifications.

void initialize_C4()

- Initializes the Origin, reads the map definitions from SD, loads the appropriate map file, and builds the desired path based on the mission files.

void test_mission

- A test method that prints a list of waypoints to cover to the destination in the final path.

void test_path()

- A test method that prints the path to destination.

void test_distance()

- A test method that prints the distance of the path completed in order to find the closest path.

void setup_C4()

- This is the setup for the C4 method. This calls the method to select the closest map, load the map and start planning the closest path. The starting node of the trike is then set to the beginning after which the planPath method is called which uses the A* algorithm to plan the path.

Libraries Used

The libraries used in the high level code is as shown below.

LEDTester

This library is used to turn the respective LED's on depending on the speed that is given and the turn direction that it receives.

void clearLights()

- This method is used to clear all the LEDs at the beginning by setting the color to 0.

void setAngles (long pos)

- This method turns the LEDs on based on the state of the trike. If the trike is turning left, then the LEDs 0 to 4 are switched on depending on how sharp of a turn, the trike has to take. Then the LEDs to the right of LED 4 are switched off.
- If the trike is just going straight, then only the LED 4 is displayed.
- If the trike is turning right, then the LEDs 4 to 8 are switched on depending on how sharp of a turn, the trike has to take. Then the LEDs to the left of LED 4 are switched off.
- This method takes in the angle at which the trike has to turn after it has been scaled down to a range of 0 to 8.

void setLeftSpeed (long sped)

- This method turns the LEDs on according to the actual speed of the trike which is received from the wheel clicks.
- The LED strip can only display a maximum value of 10 km/hour so any value beyond this is equivalent to a value of 10 km/hour.
- Depending on the value that is the passed in the corresponding LED from 1 to 10 are lit.
- This method takes in the actual speed of the trike.

void setRightSpeed (long sped)

- This method turns the LEDs on according to the desired speed at which the trike has to move according to the value that the lower level system receives from C3.
- The LED strip can only display a maximum value of 10 km/hour so any value beyond this is equivalent to a value of 10 km/hour.
- Depending on the value that is the passed in the corresponding LED from 1 to 10 are lit.
- The method takes in the desired speed at which the trike has to move.

Serial Communication

This library is a workaround for the issues with the ElcanoSerial library. While testing the communication between the lower and the higher level system, it was observed that the ElcanoSerial library would mess up if both the systems were trying to send and receive data at the same time with differing number of digits. **Note: ElcanoSerial works well when only one system tries to send or receive data at a time.**

long sendData (long number_Sent)

- In order to match the number of digits being received with the number of digits being sent, any value that has to be sent is passed to this method before it is sent to other systems.
- In the 5 digit number generated, the first number represents the number of filler values that will be placed to generate the 5 digit sequence (i.e, 9 in this case). This is then followed by the actual filler values.
- This method adds random characters in front of the actual value to be sent in order to make the value sent equal to five characters in total. The random character used in this particular method as a filler is the number 9.
- This method takes in the value sent as the parameter.

long receiveData (long number_Received)

- This method separates the original value from the value with random characters added in front (i.e, number 9 in this case) by the sendData method.
- This method takes in the value received as the parameter.

int scaleDownAngle (long angle)

- Since the LED display used as a testing interface can only display a maximum of 9 LEDs, the direction that the trike has to take is scaled down when it gets displayed on the LEDs.
- This method scales down the angle to a range between 0 to 8.
- This method takes in the angle at which the trike has to turn.

Elcano Serial

This library is used to enable the high level system communicate to the lower level system using serial UART communication. This library was reverted back to the first version of itself since the checksum was implemented incorrectly. This current version of the library is implemented as an asynchronous finite state machine. **Therefore, the "checksum" feature needs to be incorporated in this current version of ElcanoSerial.h library.**

IODue

This library holds the pin mappings to corresponding specific signals. As the name suggests this mainly is for the C4,C3 and C6 system pin usage.

Adafruit_NeoPixel

This library is incorporated in the high level system for the test interface. This library holds the information required to operate the LED strips on the testing stick wheel.

FusionData
This library is used to calculate the slope of a segment between two points and then uses it to find the fuzzy cross point average which is used in C6.

Common
This library holds the structure defined for waypoints and contains methods to acquire the correct data from GPS in addition to reading data from the buffer after parsing it.

Adafruit_GPS-master
This is the library used for the GPS module which uses the MTK33x9 chipset.

Adafruit_LSM303DLHC-master
This is the libarary used for the adafruit LSM303 compass.
Note: This library was modified to use "wire1" instead of "wire" for the I2C communication. This is an issue that was discovered with the Arduino Due. Therefore, "wire1" must be used along with SCL1 and SDA1 (physical connection) for any I2C communication.

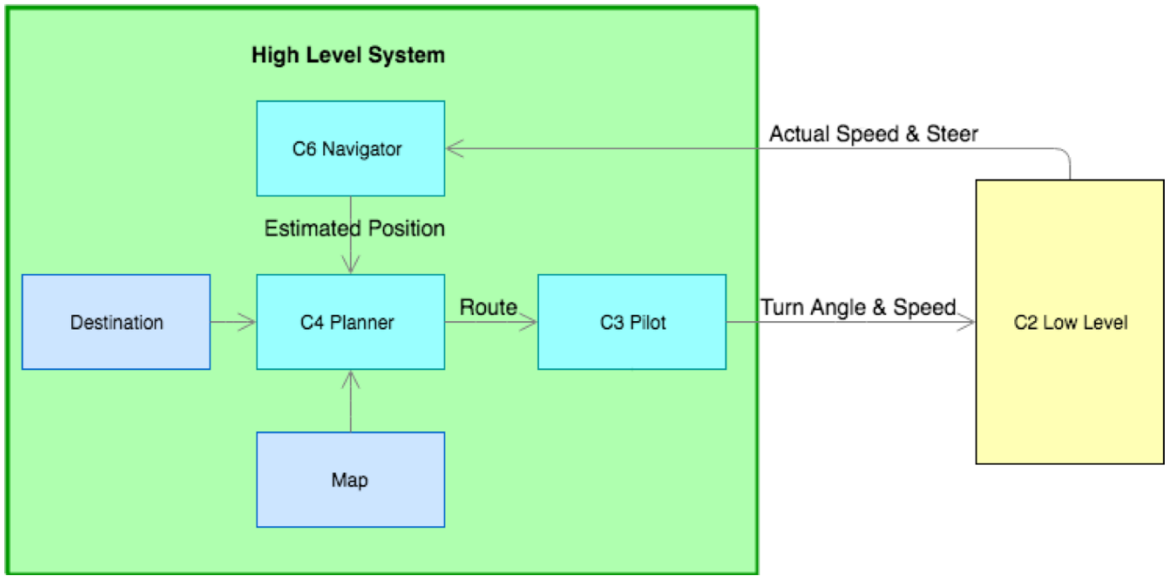
Adafruit_L3GD20_U-master
This is the library used for the gyro.

Adafruit_Sensor-master
This is the library for the Adafruit unified sensor driver. Accelerometer, magnetometer and the gyro are based on this sensor.

FastCRC-master
This library performs checksum calculation for memory blocks, strings, blobs, streaming data, and files in their application.

Scheduling System

All the three systems, i.e, C6 localization, C4 mapping and C3 pilot are all implemented in Arduino Due. Therefore, a scheduling system is designed to give each system a chance to run entirely. The scheduling system for this High Level System Re-design runs C6 completely first, then C4 but only one time through until a new destination is picked and then runs C3 completely. This process is then repeated. However, the C4 loop is not a part of this scheduling system as it is for future development where it would re-compute the path if you're too far off the track.

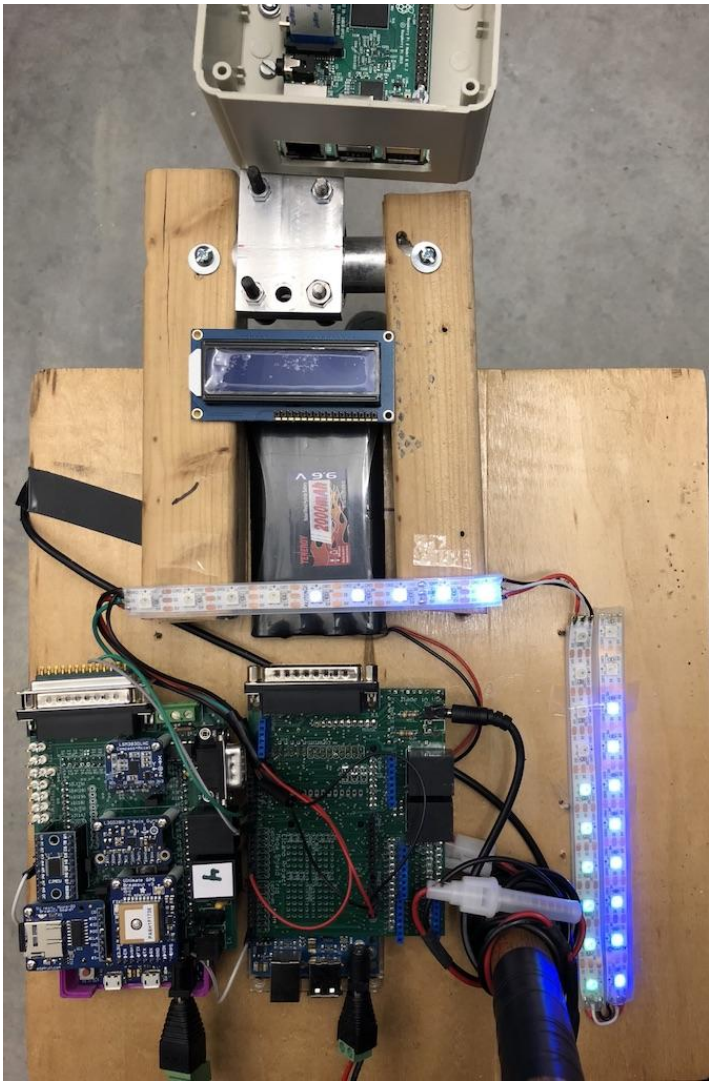


In the diagram shown above, C3 passes the desired speed of the vehicle along with the turning direction of the vehicle to the lower level system (i.e, C2). The lower level makes the trike move faster or slower depending on the desired speed and takes a turn depending on whether or not it is required. This is dependent on the turn direction received from C3. The lower level system (C2) then passes the actual speed of the trike and the wheel angle to the localization (C6). The C6 then uses this speed in getting an updated estimate of the current position using dead reckoning and data from the gps. This updated current position is then passed onto the C4 (Mapping) which re-computes the path. The C4 only re-computes the path if the destination changes and hence is called only once in setup. The updated current position is also passed onto C3 which uses it to compute the turning radius of the trike which in turn is used to find the state of the trike and the loop continues.

Test Interface

The test rig used for this project was a stick wheel with three LED strips mounted on it. The aim of this testing system was to show that the high level system is able to communicate with the lower level system based on correct current estimated position. The LED strip on the center displays the direction that the trike has to take based on whether it enters into an intersection or not. The LED strip to the right, shows the desired speed that is passed from the C3 system onto the lower level system (i.e, C2). The desired speed is calculated based on whether the trike is accelerating or decelerating which is dependent on the state of the trike at that point in time. The LED to the left of the LED strip that shows the desired speed displays the actual speed of the trike. The actual speed of the trike comes from the wheel clicks detected by the reed switch near the rear end of the wheel. Both the desired and actual speed is displayed in km/hour.

The turn direction (center LED strip) is numbered from 0 to 8 going from left to right of the strip. The LED strip turns LED 4 on if the trike's state is "straight". LED's 0 to 4 are turned on if the trike has to turn left. The LED's 0 to 4 are displayed from highest brightness to lowest brightness with the lowest brightness being at LED 4. The number of LEDs that are turned on to the left of LED 4 corresponds to the angle at which the trike should make the left turn. For example, if LEDs 4,3 and 2 are lit, then the trike needs to make a smaller degree left turn compared to when all the LEDs from 0 to 4 are lit. In a like manner, LEDs 4 to 8 are used to show that trike needs to make a right turn. The LEDs 4 to 8 are displayed from lower brightness to higher brightness with the lowest brightness LED starting from LED 4. As in the previous case, the number of LEDs lit corresponds to the angle at which the trike needs to make a turn. For example, if LEDs 4 to 8 are all lit, this means that the trike needs to take a sharp turn(i.e, turning angle is large) than if only the LEDs 4,5, and 6 are lit. The LED displaying the speeds and turn direction is shown below.



The LED display is physically connected to the digital pin 4 on the Arduino Mega connected to the lower level board. A 5 V power supply is required to operate the LED strips without damaging it. The LED strip has two ground connections.

Final Note:
Gyro needs to be implemented.
Compass has to be leveled, so might need to be calibrated.