ELEC4010M Project 1 Report

Implementation of Real-time 3D Orientation Mapping

DU Donghong, LI Junrong

***Abstract*— This project implements a system which can perform a mapping between a real object and the 3D model in Matlab in real time. It can synchronize the rotation of the object as well as the model while showing some relevant data.**

I. INTRODUCTION

In this project, the aim is to implement a real-time 3D orientation mapping model in Matlab. It is composed of an Arduino and a 9-axis IMU and a computer running the Matlab source code for this project. Its working principle is to perform a synchronized motion in the computer as the real item does. In other words, when we rotate our objects which is attached to the Arduino and IMU, in our hands, the model on the screen should perform almost the same motion.

II. PREPARATION STEPS

A. Model Drawing

In order to map the real object to a Matlab-based model, we need to draw a model in Matlab. There are some easy and convenient structures for us to composite our models in Matlab.

Our model is designed to be a mushroom. It consists of two cylinders with some additional transformations as well as rotations.

B. GUI implementation

A graphical user interface (GUI) is easier for user to learn about and play around with our project. It looks better than a simple terminal. In Matlab, there are already libraries for building up the GUI. In our project, two pop-up windows are built to perform relevant information, one for real-time motion mapping, the other one for attitude indicator.

Some data including the pitch, roll, heading and linear acceleration on three axes are also printed and updated timely.

There are also some small but useful functions implemented in our GUI.

Enable/Disable certain axes: we can enable/disable the activities on a designated axis with ticking/unticking the checkbox. This is realized by simply add some controls over the rotation angle. If we disable one axis, we simply set the angle to be 0.

Reset: we can reset the model to its default position. This function can help us to adjust out model and tune the rotation direction. This is realized by setting all degrees to 0.

III. MATH

The mathematical part of this project is mainly about the Euler rotation matrix.

A. Data from Arduino

To be noticed, the data obtained from Arduino module are mostly the velocity of that parameter. For instance, the angle value from gyroscope is degree per second. However, some other useful data like heading and rolling, is the instant value in degree.

B. Corporation with Matlab

- When dealing with the data from gyroscope, time need to be carefully considered. Before the execution of the instruction to get data, time should be recorded as well as the complement of the particular instruction. Besides, what this means is that data related instruction should be alone between the instruction of time counting for the accuracy of angles.
- For another type of data, which is the instance value. Our group have two ways to deal with them. One is to sample them several times and set the final value to the average. The other method is just to sample the value at the moment and do nothing. After comparison of the methods, the second proved to be better. The reason behind from our view is that the I/O related operations need relatively long time, resulting in larger mistakes.

IV. CORE PROCEDURES

The core procedures of this project focus on the collection of data and process of the data. After the mathematical background of the previous part, we have already made theoretical preparation for the motion. In order to realize the mapping motion, we need to get the data in IMU which indicates the acceleration, heading, pitch, roll and angular velocity of the real object. Then we can compute the rotational angle respectively.

A. Build up connections between Arduino and Computer

First, we need to make sure our Arduino and IMU work well together. Thus, we may get the required data from IMU. It is a relatively easy step to get the data. IMU has already had the required data stored inside and we just need to get them out.

There is also an existing library for IMU which is called "NAxisMotion.h". In this project we use this library. By using the Arduino Create online editor, we first setup the Arduino and then run the loop to get the data.

For the control of Arduino and IMU, we set four modes----"G", "H", "A", "R". The IMU will switch to the certain mode when we input the instructions.

And in our code, G stands for Gyro information, H stands for heading direction, A stands for acceleration and P stands for pitch as well as roll information. It can output the corresponding information in one mode.

And in the Matlab, the program will continuously send instructions to Arduino to control its mode and get the response. The responses are the data and Matlab program will record them for further use.

This is how we collect required data from IMU.

B. *Getting the essential data*

After building up the connections, we may now try to read the data in IMU and turn them into our Matlab program.

As for the rotation angle, we compute them from multiplying the angular velocity by a short time interval. This approximate the real angle the object rotates because it is a very short time interval and the moved angle in every interval is relatively small. So here we need to read the angular velocity from the IMU. The corresponding functions are readGyroX(), readGyroY(), readGyroZ().

As for the heading parameters, there is already an existing function called readEulerHeading() in the library. So we can simply use this function to read out the heading data.

The pitch and roll are also existing data in IMU and we may use readEulerPitch() and readEulerRoll() to access them. They are helpful when we are manipulating the attitude indicator.

We also record the linear acceleration in our code for possible usage later. To get them, the functions are readLinearAccelX(), readLinearAccelY(), readLinearAccelZ().

C. *Displaying on the screen*

We have now prepared the 3D model in Matlab and get access to all required data. The final step is to plug in the data and apply the formula onto the model.

With the rotation matrix function, we can control our model by plugging in the rotation degree in real time and performing rotational motion on the model.

For the attitude indicator, we still use the same model but with the pitch and roll data.

GUI part also print the data on the screen with heading and acceleration on x, y and z axis.

V. Conclusion

This project is divided into three major parts. First part is to prepare the 3D model for further use. Second part is to compute the rotation matrix using the accessible data. Third part is to plug in our data and perform the real-time mapping rotations on the screen.

Appendix

1: Source code list and README:

euler_angle.cpp: build up connections between the Arduino and web editor on PC. It sets up the Arduino to response to different modes instructions.closeSerial.m: this function just clears up the workspace and terminates the processes.

dragzoom.m: this includes the function to perform drag and zoom.

getAngle.m: do the difference of the previous angle and the current angle. Then we can get the delta angle which is required for rotation.

readGyro.m: send instructions to Arduino and get gyro as reply.

setupSerial.m: set up the Arduino for sending and receiving serials

trial_angle.m: this function serves as a converter from the Arduino to get the angular velocity and time, then we compute the degrees we have turned.

trial_euler.m: this function gets the roll and pitch from Arduino.

trial_heading.m: this function gets the heading information from the Arduino.

trial_velocity.m: this function gets the acceleration information from the Arduino.

trial.m: this serves as the main function of performing the basic real-time rotation mapping of objects.

trial_two.m: this function performs the pitch and roll functions. It builds up the model and rotate the model as attitude indicator.

trial_v.m: this is the main function for perform the linear acceleration data on the screen read from the IMU.

stlread.m: convert a stl file into points and face matrices.

assem.stl: the stl file for our onshape model.

2: A demo video for the functionalities of this project. The demonstration video shows our functionalities of rotation mapping, attitude indicator, drag and zoom functions and the acceleration indicator.

3: Demonstration video link on YouTube: https://www.youtube.com/watch?v=KSwNbQn4K8Q

References

[1] https://github.com/arduino-org/arduino-library-nine-axes-motion
(the nine axis library for Arduino)

[2] https://cn.mathworks.com/help/matlab/ref/hgtransform.html
(hgtransform function help)

[3] https://cn.mathworks.com/help/matlab/ref/rotate.html
(rotate function example in Matlab)

[4] https://cn.mathworks.com/help/matlab/ref/uicontrol.html?searchHighlight=uicontrol&s_tid=doc_srchtitle
(library for making the GUI)

[5] https://cn.mathworks.com/matlabcentral/fileexchange/29276-dragzoom-drag-and-zoom-tool
(drag and zoom functionalities library)

euler_angle.cpp

```
#include "NAxisMotion.h"        //Contains the bridge code between the API and the Arduino Environment
#include <Wire.h>
#include <Servo.h>

NAxisMotion mySensor;         //Object that for the sensor
unsigned long lastStreamTime = 0;     //To store the last streamed time stamp
const int streamPeriod = 20;           //To stream at 50Hz without using additional timers (time period(ms) =1000/frequency(Hz))


void setup() //This code is executed once
{
  //Peripheral Initialization
  Wire.begin();
  Serial.begin(115200);                  //Initialize the Serial Port to view information on the Serial Monitor
  I2C.begin(105);                //Initialize I2C communication to the let the library communicate with the sensor.
  //myservo.attach(9);
  //myservo.write(0);
  //Sensor Initialization
  mySensor.initSensor();          //The I2C Address can be changed here inside this function in the library
  mySensor.setOperationMode(OPERATION_MODE_NDOF);   //Can be configured to other operation modes as desired
  mySensor.setUpdateMode(MANUAL); //The default is AUTO. Changing to MANUAL requires calling the relevant update functions prior to calling the read functions

  delay(1500);
  Serial.println('a');
  char a = 'b';
  while(a!='a')
  {
    a=Serial.read();
  }
}

void loop() //This code is looped forever
{

    if ((millis() - lastStreamTime) >= streamPeriod){
     lastStreamTime = millis();
    int mode = Serial.read();
    if(mode=='G'){
       mySensor.updateGyro();

       //mySensor.updateGravAccel();
       Serial.println((mySensor.readGyroX()));
       Serial.println((mySensor.readGyroY()));
       Serial.println((mySensor.readGyroZ()));
       //Serial.println((mySensor.readAccelZ()));
       delay(50);
    }
    if(mode=='H'){
      mySensor.updateEuler();
      Serial.println((mySensor.readEulerHeading()));
      delay(50);
    }
    if(mode=='A'){
       mySensor.updateLinearAccel();
       Serial.println((mySensor.readLinearAccelX()));
       Serial.println((mySensor.readLinearAccelY()));
       Serial.println((mySensor.readLinearAccelZ()));
       delay(20);
    }
    if(mode=='F'){
       mySensor.updateEuler();
       Serial.println((mySensor.readEulerRoll()));
       Serial.println((mySensor.readEulerPitch()));
       mySensor.updateGyro();
       Serial.println((mySensor.readGyroZ()));
       delay(50);
    }
    if(mode=='R'){
       mySensor.updateEuler();
       Serial.println((mySensor.readEulerRoll()));
       Serial.println((mySensor.readEulerPitch()));
       //for roll: left down is positive, range is [-108,180] in degree.
       //for pitch: head down is positive, range is [-180,180] with unit of degree.
       //base on the earth horizon.
       delay(50);
    }
    }
    //Update the Euler data into the structure of the object
    //mySensor.updateCalibStatus();
    //Update the Calibration Status
}

```