CAPSTONE PROJECT FINAL REPORT

Augmented Reality with Location Tracking

Authors:
Drew Barclay
Maricar Aliasut
Llandro Ojeda

Advisors:
Dr. Robert MacLeod
Dr. Ekram Hossain

Final report submitted in partial satisfaction of the requirements for the degree of Bachelor of Science

in

Electrical & Computer Engineering

in the

Faculty of Engineering

of the

University of Manitoba

Spring 2017

University of Manitoba

Abstract

Faculty of Engineering
Electrical & Computer Engineering

Bachelor of Science

Augmented Reality with Location Tracking

by Drew Barclay
Maricar Aliasut
Llandro Ojeda

- Statement of the problem
  - Procedures and methods used
  - Results and Conclusions

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor...

Contributions

The contribution page gives a concise statement of what novelty the thesis makes. In a multi-author thesis, it also gives the attribution for each component of the work, including the report.

Contents

List of Figures

List of Tables

List of Abbreviations

LAH	**L**ist **A**bbreviations **H**ere
WSF	**W**hat (it) **S**tands **F**or

Physical Constants

Speed of Light $c_0 = 2.997\,924\,58 \times 10^8\,\mathrm{m\,s^{-1}}$ (exact)

List of Symbols

a	distance	m
P	power	W ($\mathrm{J\,s^{-1}}$)
ω	angular frequency	rad

For/Dedicated to/To my…

Chapter 1

Introduction

1.1 Stuff

The Introduction can be made into its own chapter but no more than about 10% of the space of the total report should be allocated to introductory remarks. The Introduction is a good place to give a literature review (i.e., if a literature review is not the main purpose of the report), put the subject matter into context, and give the relevant motivation for the work which was performed. The remaining body of the report should be broken up into chapters in a logical manner.

1.1.1 Motivation

Talk about FPS games where positions are shown and allow easy finding of team members?

1.1.2 Overview of the Design

Go over tags/anchors (define each), how each tag is hooked up to a cellphone by USB, where cellphone calculates positions and shows them. Talk specifically about Arduino/DWM1000?

Chapter 2

Rangefinding

2.1 Overview

Talk about the goal of this section: to get a bunch of devices determining their range to each other wirelessly.

Go over the setup of tags/anchors once more, explain the hardware we're using for each.

Talk about how distance is determined via time of flight by calculating speed of light and delay.

Table of requirements (low power, good range, work indoors).

2.2 Math

Go over time of flight.

2.3 Bluetooth - A Failed Approach

Talk about how originally we planned on doing this via Bluetooth, but Android did not provide tight enough guarantees. This motivated the use of Arduino and specialty chips. The DWM1000 is, of course, fantastic at it.

2.4 Ultrawideband

A bit on this would be good! Go into the DWM1000's features.

2.5 Design of the Hardware

Go into PCB design, talk about difficulties and options of trying to condense the size of the tag down. Show off anchor breadboard design. Talk briefly about FTDI.

2.6 Arduino Software

Go into our open source library to help with the use of the DWM by thotro. Talk about the protocol we made.

2.7 DWM1000 Bugs

Go into specific issues with calibrating antennae delay, with sending delayed transmissions, etc.

2.8 Results

Go into how accurate rangefinding is, operating frequency, etc. Have tables!

2.9 Conclusion

Talk about how we now have a subsystem which can fully handle calculating ranges between things.

Chapter 3

Position Calculation

3.1 Overview

Talk about the goal of this section: to calculate positions of objects given the ranges between them. Talk about how Arduinos and Androids are connected by FTDI.

3.2 High-Level Example in 2D

Maybe this?

3.3 Extending This to 3D

Do the math on how we actually calculate things here.

3.4 Arbitrariness (pick a better title later)

The positions calculated from ranges do not correspond to the real world! We must use the cellphones' accelerometer (for gravity) and magnetic sensor (for compass direction/inclination) to map these positions to reality.

Go over how this is done.

3.5 Getting the Ranges

Talk about FTDI and our protocol to communicate from Arduino to Android. Go over how they are parsed (put in appendix of parsing code)

3.6 Conclusion

This subsystem calculates positions; we figured out how ranges are obtained, how they are calculated. Go over this in more detail.

Chapter 4

Augmented Reality

4.1 Overview

Talk about the goal of this section: to take positions from the position calculation subsystem, then render them on the screen to show where things are (even behind walls, etc.)

Make sure to note that we'll have youtube videos up and give links.

Don't forget an appendix with major code.

4.2 Tech Used

Go over OpenGL, the 3D text rendering library we're using, Android's rotation calculations.

4.3 3D Math Overview

Touch on matrices, projection/view/model matrices. Talk about how we use OpenGL's implementation to place things in 3D space.

4.4 Camera Field of View and OpenGL Field of View

Talk about how we make positions accurate here. Go over how we just overlay the camera and OpenGL surface on each other. Discuss Android camera limitations/FPS results.

4.5 Cellphone Rotation

Discuss in detail more about how Android implements getting the rotation, the limits of it when moving + in strong magnetic fields.

4.6 Billboard Effect

Talk about how doing the inverted view matrix multiplication causes things to always face the screen. Give pictures! Motivation here is to make 2D images that we can place in 3D space and have them face the screen.

4.7 HUD

Go over how the HUD is made. (Still need to finish that too so we can get pictures.)

4.8 Calibration

Talk about how we can take the cellphone's rotation matrix and calibrate the positions we calculate from anchors/tags.

4.9 Results

Show off how accurate we are. Have Youtube videos displaying such.

4.10 Conclusion

Conclusion of this section: we can render positions on the screen and have 3D math to do so etc.

Chapter 5

Conclusion

5.1 Stuff

The final chapter is the Conclusions chapter; there should be no surprises in the Conclusion: it is just a summary–about a page long–of what has been achieved. A short description of possible future work can be included in this chapter.

Appendix A

Arduino Code

```
//Created by Drew Barclay
//Code uses thotro's dwm1000-arduino project
//Protocol: one byte for ID, then five bytes for the timestamp of the sending
//For each device, append one byte for the ID of the device, one byte for the

#include <SPI.h>
#include <DW1000.h>

class Device  {
public:
  byte id;
  byte transmissionCount;
  DW1000Time timeDevicePrevSent;
  DW1000Time timePrevReceived;
  DW1000Time timeSent;
  DW1000Time timeDeviceReceived;
  DW1000Time timeDeviceSent;
  DW1000Time timeReceived;
  float lastComputedRange;
  bool hasReplied;

  Device() : lastComputedRange(0.0f), id(0), hasReplied(false) {}

  void computeRange();

  float getLastComputedRange() {
    return this->lastComputedRange;
  }
};

// CONSTANTS AND DATA START
//number of devices that can form a network at once
#define NUM_DEVICES 6
Device devices[NUM_DEVICES];
int curNumDevices;

// connection pins
const uint8_t PIN_RST = 9; // reset pin
const uint8_t PIN_IRQ = 2; // irq pin
const uint8_t PIN_SS = SS; // spi select pin
```

```
// data buffer
#define LEN_DATA 256
byte data[LEN_DATA];

//id for this device
const byte OUR_ID = 5;

long lastTransmission; //from millis()

// delay time before sending a message, should be at least 3ms (3000us)
const unsigned int DELAY_TIME_US = 2048 + 1000 + NUM_DEVICES*83 + 200; //shoul

volatile bool received; //Set when we are interrupted because we have received
// CONSTANTS AND DATA END

void Device::computeRange() {
  // only call this when timestamps are correct, otherwise strangeness may res
  // asymmetric two-way ranging (more computationly intense, less error prone)
  DW1000Time round1 = (timeDeviceReceived - timeDevicePrevSent).wrap();
  DW1000Time reply1 = (timeSent - timePrevReceived).wrap();
  DW1000Time round2 = (timeReceived - timeSent).wrap();
  DW1000Time reply2 = (timeDeviceSent - timeDeviceReceived).wrap();

  DW1000Time tof = (round1 * round2 - reply1 * reply2) / (round1 + round2 + re
  this->lastComputedRange = tof.getAsMeters();
}

void setup() {
    received = false;
    curNumDevices = 0;
    lastTransmission = millis();

    Serial.begin(115200);
    delay(1000);
    // initialize the driver
    DW1000.begin(PIN_IRQ, PIN_RST);
    DW1000.select(PIN_SS);
    Serial.println(F("DW1000␣initialized␣..."));
    // general configuration
    DW1000.newConfiguration();
    DW1000.setDefaults();
    DW1000.setDeviceAddress(OUR_ID);
    DW1000.setNetworkId(10);
    DW1000.enableMode(DW1000.MODE_LONGDATA_RANGE_ACCURACY);
    DW1000.commitConfiguration();
    Serial.println(F("Committed␣configuration␣..."));

    // attach callback for (successfully) sent and received messages
    DW1000.attachSentHandler(handleSent);
    DW1000.attachReceivedHandler(handleReceived);
```

```
    DW1000.attachErrorHandler(handleError);
    DW1000.attachReceiveFailedHandler(handleReceiveFailed);

    receiver(); //start receiving
}

void handleError() {
  Serial.println("Error!");
}

void handleReceiveFailed() {
  Serial.println("Receive failed!");
}

void handleSent() {

}

void handleReceived() {
  received = true;
}

void receiver() {
    DW1000.newReceive();
    DW1000.setDefaults();
    // so we don't need to restart the receiver manually
    DW1000.receivePermanently(true);
    DW1000.startReceive();
}

void parseReceived() {
  unsigned int len = DW1000.getDataLength();
  DW1000Time timeReceived;
  DW1000.getData(data, len);
  DW1000.getReceiveTimestamp(timeReceived);

  if (len < 6) {
     Serial.println("Received message with length <6, error.");
     return;
  }

  //Parse data
  //First byte, ID
  byte fromID = data[0];
  //Second byte, 5 byte timestamp when the transmission was sent (their clock)
  DW1000Time timeDeviceSent(data + 1);

  //If this device is not in our list, add it now.
  int idx = -1;
  for (int i = 0; i < curNumDevices; i++) {
   if (devices[i].id == fromID) {
```

```
      idx = i;
    }
  }
  if (idx == -1) { //If we haven't seen this device before...
    Serial.print("New device found. ID: "); Serial.println(fromID);
    if (curNumDevices == NUM_DEVICES) {
      Serial.println("Max # of devices exceeded. Returning early from receive.
      return;
    }
    devices[curNumDevices].id = fromID;
    devices[curNumDevices].timeDeviceSent = timeDeviceSent;
    devices[curNumDevices].transmissionCount = 1;
    idx = curNumDevices;
    curNumDevices++;
  }

  devices[idx].hasReplied = true;

  //Now, a list of device-specific stuff
  for (int i = 6; i < len;) {
    //First byte, device ID
    byte deviceID = data[i];
    i++;

    //Second byte, transmission counter
    byte transmissionCount = data[i];
    i++;

    //Next five bytes are the timestamp of when the device received our last t
    DW1000Time timeDeviceReceived(data + i);
    i += 5;

    //Next four bytes are a float representing the last calculated range
    float range;
    memcpy(&range, data + i, 4);
    i+= 4;

    //Is this our device? If so, we have an update to do and a range to report
    if (deviceID == OUR_ID) {
      //Mark down the two timestamps it included, as well as the time we recei
      devices[idx].timeDeviceReceived = timeDeviceReceived;
      devices[idx].timeDeviceSent = timeDeviceSent;
      devices[idx].timeReceived = timeReceived;

      Serial.print("Transmission received from tag "); Serial.print(devices[id

      //If everything looks good, we can compute the range!
      if (transmissionCount == 0) {
        //Error sending, reset everything.
        devices[idx].transmissionCount = 1;
      } else if (devices[idx].transmissionCount == transmissionCount) {
```

```
        if (devices[idx].transmissionCount > 1) {
          devices[idx].computeRange();
          Serial.print("!range "); Serial.print(OUR_ID); Serial.print(" "); Se
        }
        devices[idx].transmissionCount++;
      } else {
        //Error in transmission!
        devices[idx].transmissionCount = 0;
        Serial.println("Transmission count does not match.");
      }

      devices[idx].timeDevicePrevSent = timeDeviceSent;
      devices[idx].timePrevReceived = timeReceived;
    }

    Serial.print("!range "); Serial.print(fromID); Serial.print(" "); Serial.p
  }
  //TODO if our device was not in the list and we think it should have been, r
}

void doTransmit() {
  data[0] = OUR_ID;

  //Normally we would set the timestamp for when we send here (starting at the

  int curByte = 6;
  for (int i = 0; i < curNumDevices; i++) {
    data[curByte] = devices[i].id;
    curByte++;
    data[curByte] = devices[i].transmissionCount;
    curByte++;
    devices[i].timeReceived.getTimestamp(data + curByte); //last timestamp wil
    curByte += 5;
    float range = devices[i].getLastComputedRange();
    memcpy(data + curByte, &range, 4); //floats are 4 bytes
    curByte += 4;

    if (devices[i].hasReplied) {
      devices[i].transmissionCount++; //Increment for every transmission, the
      devices[i].hasReplied = false; //Set to not for this round
    }
  }

  //Do the actual transmission
  DW1000.newTransmit();
  DW1000.setDefaults();

  //Now we figure out the time to send this message!
  DW1000Time deltaTime = DW1000Time(DELAY_TIME_US, DW1000Time::MICROSECONDS);
  DW1000Time timeSent = DW1000.setDelay(deltaTime);
  timeSent.getTimestamp(data + 1); //set second byte (5 bytes will be written)
```

```
  DW1000.setData(data, curByte);
  DW1000.startTransmit();

  for (int i = 0; i < NUM_DEVICES; i++) {
    devices[i].timeSent = timeSent;
  }
}

void loop() {
  unsigned long curMillis = millis();

  if (received) {
    received = false;
    parseReceived();
  }

  if (curMillis - lastTransmission > 300) {
    doTransmit();
    lastTransmission = curMillis;
  }
}
```