Lighterface - valosensoreiden soveltuvuus osoittimen ohjaukseen

Martin Yrjölä - 84086N

12. marraskuuta 2015

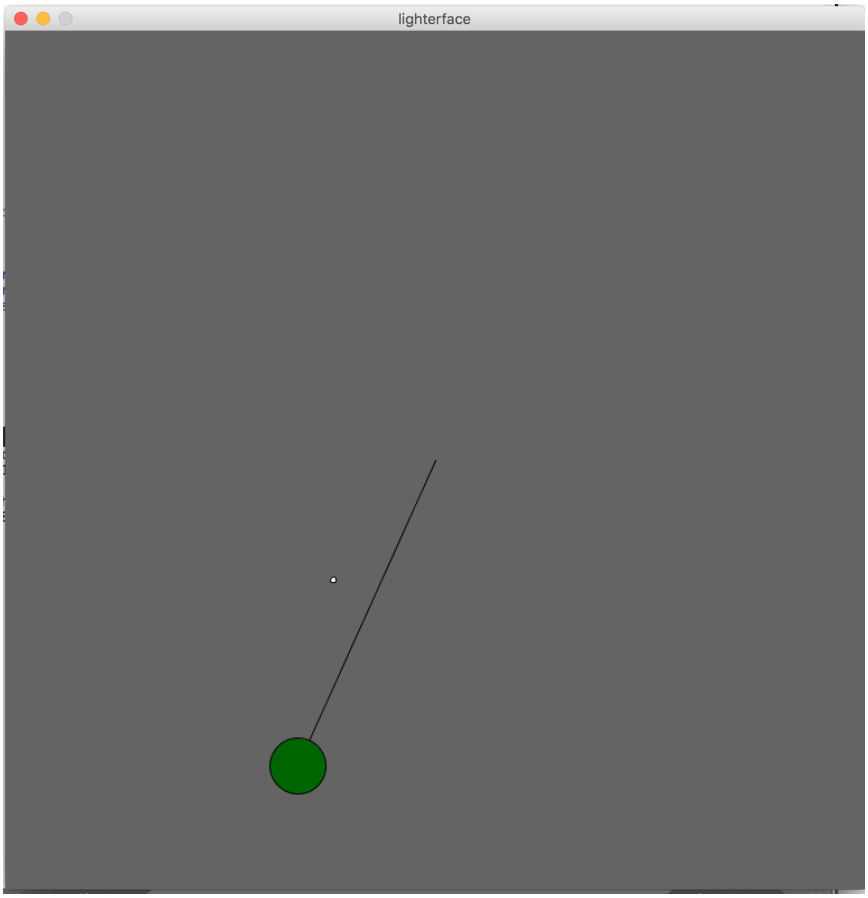
Sisältö

1 Johdanto

Hiiri on vallitseva tapa osoittimen ohjaukseen. Tässä tutkimuksessa selvitetään vaihtoehtoista ohjaustapaa käyttäen kahta valosensoria.

2 Metodiikka

Mittaukseen käytin Processing ohjelmointikielellä toteutettua ohjelmaa nimeltä Lighterface. Olen taltioinut demovideon osoitteeseen `https://youtu.be/LvIR9VB5Kk4`.



Kuva 1: Kuvakaappaus Lighterface ohjelmasta

Lighterface piirtää ruudun keskelle ympyrän. Kun osoitin on leijunut ympyrän sisällä kolme sekuntia se häviää ja uusi vihreä ympyrä esiintyy 300 pikselin päähän satunnaiseen suuntaan. Käyttäjän kuuluu viedä osoitin vihreän ympyrän sisälle niin nopeasti ja tarkasti kuin mahdollista. Tällä tavalla voimme mitata kuinka hyvin eri ohjaustavat soveltuvat suorien viivojen piirtämiseen.

Suoritukset tallennetaan CSV-tiedostoon. Tiedostoon tallennetaan jokaisen piirtokutsun yhteydessä xy-koordinaatit sekä kuinka monta millisekuntia on kulunut viivan piirtämisen alkamisesta. Tiedoston viimeinen rivi sisältää viivan päätepisteen ja piirron kokonaisajan.

```
x,y,milliseconds
384,385,16
393,345,32
...
559,140,1016
```

Kuva 2: Lyhennetty esimerkki mittaustiedostosta.

Koska viivaa piirretään useaan suuntaan on järkevää normalisoida data muuttamalla koordinaatistoa niin, että x-akseli on viivan suuntainen.

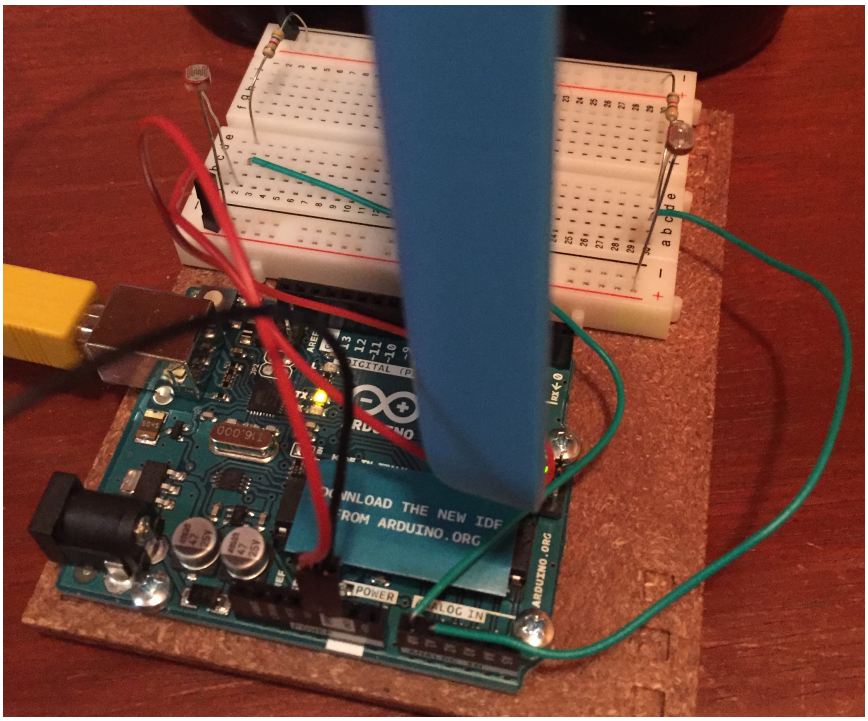
```
normalizeData <- function(filename) {
  data = read.csv(filename)
  xy <- data[1:2]
  origin <-  xy[1,]
  p <- tail(xy, n=1) - origin
  angle <- -atan2(p$y, p$x)

  xy.normalized <- within(xy, {
    x <- x - origin$x
    y <- y - origin$y
  })

  xy.rotated <- within(xy.normalized, {
    x.rotated <- x*cos(angle) - y * sin(angle)
    y.rotated <- x*sin(angle) + y * cos(angle)
    x <- x.rotated
    y <- y.rotated
  })
  return(xy.rotated[1:2])
}
```

Kuva 3: Mittausdatan normalisointifunktio

Näin voimme siis ryhmittää mittausdatat ja saada verrannollisia kuvaajia. Toinen hyöty on, että y-akseli kertoo etäisyyden viivasta, jota voimme käyttää eri ohjausmenetelmien tarkkuuden mittaamiseen.

3 Koeasetelma

Tässä tutkimuksessa vertailemme eri ohjaustapoja valosensoreilla. Rakensin kahta valosensoria mittaavaan järjestelmän käyttäen Arduinoa. Kuva 4 näyttää miten kytkennät on tehty.



Kuva 4: Koejärjestelmä, näkösuoja sensoreiden välissä häiriöiden välttämiseksi.

Ensimmäiseksi mittasin ohjausta varjostamalla valosensoreita käsilläni valaistussa huoneessa. Tätä ohjaustapaa kuvaan nimellä Varjo. Tämä ohjaustapa tuntui epäluonnolliselta, mutta pienen harjoittelun jälkeen sain osoittimen pisteestä A pisteeseen B, mutta ei välttämättä suorassa viivassa.

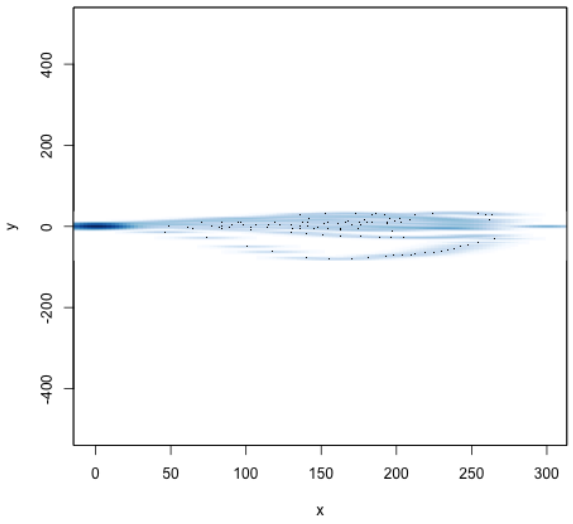
Toinen ohjaustapa oli pimeässä huoneessa ohjaus käyttäen kahta taskulamppua. Kutsun tätä ohjaustapaa nimellä Valo. Nopean kokeilun jälkeen huomasin, että parhaaseen tulokseen pääsee, kun taskulamppuja liikuttaa kiertoliikeellä, täten ohjaustapa muistuttaa nuppien vääntämistä. Tämä tuntui luonnolisemmalta tavalta ohjata osoitinta, mutta ohjaustavan toteutus vaatii enemmän välineitä.

Otin myös näytteitä hiiriohjauksella vertailuarvoiksi.

4 Tulokset

Käytämme N:nä viivojen piirtojen määrä. Keräsin 13 hiirellä piirrettyä viivaa vertailuarvoiksi. Varjo ohjaustavalla piirsin 23 viivaa ja Valo ohjaustavalla 11 viivaa.

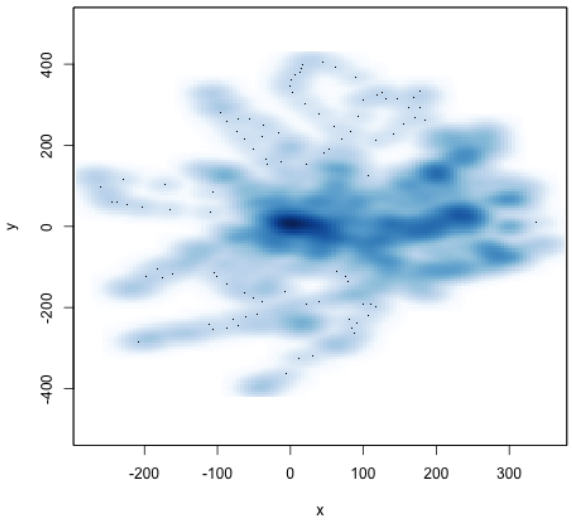
Kuvissa 5, 6 ja 7 näemme tiheyskuvaajat eri ohjaustapojen välillä. Silmämääräisesti tarkasteltuna näemme, että hiiriohjaus on tarkin tapa piirtää viiva. Varjo ohjauksessa osoitin heittelehti paljon ruudulla, joten osoitin eksyy kauas piirettävältä viivalta. Valo ohjaustavalla on hieman parempi tulos verrattuna Varjo ohjaukseen.



Kuva 5: Hiiriohjauksen tiheyskuvaaja

Taulukossa 1 on kerätty deskriptiivistä statistiikka etäisyydestä piirrettävään viivaan. Tilastot vahvistavat oletukset eri ohjaustapojen tarkkuuksista. Niin keskimääräinen virhe kuin standardipoikkeaman suuruus antaa meille paremmuusjärjestyksen Hiiri, Valo ja Varjo.

Mittasin myös aikaa viivan piirtämiseen millisekunneissa. Taulukkoon 2 on



Kuva 6: Varjo-ohjauksen tiheyskuvaaja

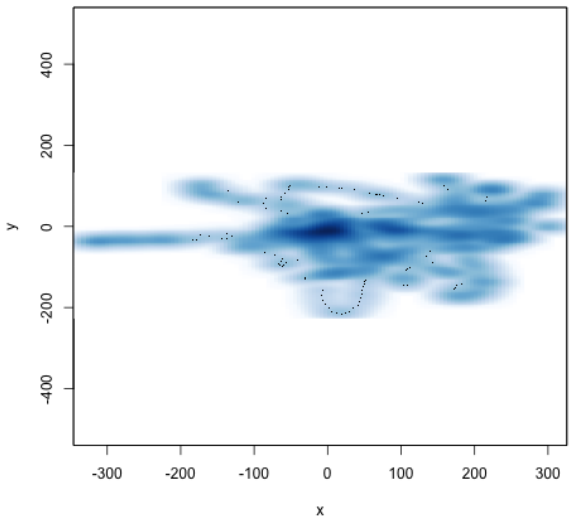
Taulukko 1: Deskriptiivistä statistiikkaa eri ohjaustapojen tarkkuudesta

	MAE	RMSE	σ
Hiiri	7.99549785832045	14.2422858883824	14.1709191594303
Varjo	62.0711795386004	91.6029119883971	91.1732261676689
Valo	42.4205676162468	57.6619883335222	54.8428730006106

kerätty tähän liittyvät tilastot. Hieman yllättäen Valo ja Varjo ovat suunnilleen yhtä nopeita ohjaustapoja mediaania tarkastellen. Varjo ohjaustavassa on runsaasti korkeampi maksimi kuin muissa ohjaustavoissa, joka nostaa keskiarvon myös hitaimmalle tasolle.

5 Johtopäätökset ja suositukset

Niin Varjo kuin Valo ohjaustavat eivät sovellu hiiren korvikkeeksi tarkkuutta vaativaan ohjaukseen. Jos haluaa toteuttaa erilaisen ohjaustavan esimerkiksi kokeelliseksi käyttöliittymäksi, eikä halua turhauttaa käyttäjää epäluonnollisella ohjauksella suosittelen Valo ohjaustapaa. Varjo ohjaustapaa kannattaa



Kuva 7: Valo-ohjauksen tiheyskuvaaja

Taulukko 2: Deskriptiivistä statistiikkaa eri ohjaustapojen nopeudesta

	Min	Max	Keskiarvo	Mediaani	σ
Hiiri	693	1350	1019.61538461538	1013	160.55971810178
Varjo	1300	9047	3334.60869565217	2816	1926.40550199141
Valo	1633	5298	3604.09090909091	3448	1227.18217511056

harkita mikäli taskulamput ja hämärä huone osoittautuvat liian hankaliksi toteutuksen kannalta.

6 Liitteet

6.1 Lighterface ohjelman koodi

```
/**
 * Lighterface - user interface research based on two photoresistors
 *
 * Two photoresistors gives a pointer's x and y coordinates. The task
```

```
 * is to write straight lines. We need a way to show this goal and
 * make sampling of the path taken when writing the line with a tricky
 * control scheme.
 */

import processing.serial.*;

import cc.arduino.*;

final int ANALOG_MAX = 1023;

Arduino arduino;

// Line to draw
PVector line;

int middleX, middleY;

int goalCircleRadius = 50;

int startDrawingLineInMillis = -1;

int minX = ANALOG_MAX;
int minY = ANALOG_MAX;
int maxX = 0;
int maxY = 0;

boolean drawLineInProgress = false;

// Change this to false when you want to control the pointer with an
// Arduino's 0 and 1 analog ports.
boolean mousePointer = true;

ArrayList<Integer> pointerXCoordinates = new ArrayList<Integer>();
ArrayList<Integer> pointerYCoordinates = new ArrayList<Integer>();
ArrayList<Integer> pointerMillis = new ArrayList<Integer>();
int startDrawingLineMillis;

int now = 0;
```

```
PrintWriter output;

void setup() {
  size(768, 768);
  middleX = width/2;
  middleY = height/2;
  newFileWithHeader();

  if (!mousePointer) {
    // Prints out the available serial ports.
    println(Arduino.list());

    // Modify this line, by changing the "0" to the index of the serial
    // port corresponding to your Arduino board (as it appears in the list
    // printed by the line above).
    arduino = new Arduino(this, Arduino.list()[5], 57600);
  }
}

void newFileWithHeader() {
  output = createWriter("results" + day() + "_" + month() + "_" + year() +
                        "_" + hour() + minute() + second() + ".csv");
  output.println("x,y,milliseconds");
  int halfWidth = width / 2;
  int halfHeight = height / 2;
  output.println(halfWidth + "," + halfHeight + ",0");
}

void draw() {
  now = millis();
  fill(100, 0.8);

  background(100);

  int x = mouseX;
  int y = mouseY;

  if (!mousePointer) {
    x = arduino.analogRead(0);
    y = arduino.analogRead(1);
```

```
  if (x > maxX) {
    maxX = x;
  } else if (x < minX) {
    minX = x;
  }
  if (y > maxY) {
    maxY = y;
  } else if (y < minY) {
    minY = y;
  }

  println("x = " + x);
  println("y = " + y);

  x = int(map(x, minX, maxX, 0, width));
  y = int(map(y, minY, maxY, 0, height));

  println("x = " + x);
  println("y = " + y);
  println("minX = " + minX);
  println("minY = " + minY);
  println("maxX = " + maxX);
  println("maxY = " + maxY);
}

// Plot the cursor
fill(255);
ellipse(x, y, 5, 5);

// Start drawing the line from the middle. Denote this region with a circle
fill(100);
if (drawLineInProgress) {
  float goalX = middleX + line.x;
  float goalY = middleY + line.y;

  // Draw the line
  line(middleX, middleY, goalX, goalY);

  // Draw goal circle
```

```
  fill(0, 100, 0);
  ellipse(goalX, goalY, goalCircleRadius, goalCircleRadius);

  pointerXCoordinates.add(x);
  pointerYCoordinates.add(y);
  pointerMillis.add(now-startDrawingLineMillis);

  if (dist(x, y, goalX, goalY) < goalCircleRadius) {
    drawLineInProgress = false;
    for (int i = 0; i < pointerXCoordinates.size(); i++) {
      output.print(pointerXCoordinates.get(i));
      output.print(",");
      output.print(pointerYCoordinates.get(i));
      output.print(",");
      output.println(pointerMillis.get(i));
    }
    output.print(int(goalX));
    output.print(",");
    output.print(int(goalY));
    output.print(",");
    output.println(now-startDrawingLineMillis);
    output.flush(); // Writes the remaining data to the file
    output.close(); // Finishes the file
    newFileWithHeader();
    pointerXCoordinates.clear();
    pointerYCoordinates.clear();
    pointerMillis.clear();
  }
} else {
  ellipse(middleX, middleY, goalCircleRadius, goalCircleRadius);
  if (dist(x, y, middleX, middleY) < goalCircleRadius) {
    fill(0, 100, 0);
    if (startDrawingLineInMillis == -1) {
      // Start drawing line in three seconds.
      startDrawingLineInMillis = now + 3000;
    } else if (startDrawingLineInMillis < now) {
      // Start drawing line.
      drawLineInProgress = true;
      startDrawingLineMillis = now;
      line = PVector.random2D();
```

```
        line.setMag(300);
      }
    } else {
      // Not anymore inside start, try again.
      startDrawingLineInMillis = -1;
    }
  }
}
```

6.2 Analyysikoodi R-kielessä

```
data.mouse <- Sys.glob("*mouse*.csv")
data.shadow <- Sys.glob("*shadow*.csv")
data.light <- Sys.glob("*light*.csv")

options(digits=5)

normalizeData <- function(filename) {
  data = read.csv(filename)
  xy <- data[1:2]
  origin <-  xy[1,]
  p <- tail(xy, n=1) - origin
  angle <- -atan2(p$y, p$x)

  xy.normalized <- within(xy, {
    x <- x - origin$x
    y <- y - origin$y
  })

  xy.rotated <- within(xy.normalized, {
    x.rotated <- x*cos(angle) - y * sin(angle)
    y.rotated <- x*sin(angle) + y * cos(angle)
    x <- x.rotated
    y <- y.rotated
  })
  return(xy.rotated[1:2])
}

groupData <- function(filenames) {
```

```
  data.normalized <- Map(normalizeData, filenames)
  return(do.call("rbind", data.normalized))
}

results.mouse <- groupData(data.mouse)
results.light <- groupData(data.light)
results.shadow <- groupData(data.shadow)

scatter.ylim = c(-500, 500)
smoothScatter(results.mouse, ylim=scatter.ylim)

smoothScatter(results.shadow, ylim=scatter.ylim)

smoothScatter(results.light, ylim=scatter.ylim)

# Descriptive statistics on the distance from the line, which happens to be the
# y-axis value after rotation
results.distance <- list(results.mouse$y, results.shadow$y, results.light$y)

MeanAbsoluteError <- function(x) {
  mean(abs(x))
}

RootMeanSquareError <- function(x) {
  sqrt(mean(x^2))
}

results.distance.descr <- data.frame(sapply(results.distance, MeanAbsoluteError),
                                     sapply(results.distance, RootMeanSquareError),
                                     sapply(results.distance, sd))

colnames(results.distance.descr) <- c("MAE", "RMSE", "\\sigma")
control.rownames <- c("Hiiri", "Varjo", "Valo")
rownames(results.distance.descr) <- control.rownames

results.distance.descr

extractTime <- function(filename) {
  data <- read.csv(filename)
  millis <- data$milliseconds
```

```
  tail(millis, n=1)
}
times <-  list(sapply(data.mouse, extractTime),
               sapply(data.shadow, extractTime),
               sapply(data.light, extractTime))

times.descr <- data.frame(sapply(times, min),
                          sapply(times, max),
                          sapply(times, mean),
                          sapply(times, median),
                          sapply(times, sd))

colnames(times.descr) <- c("Min", "Max", "Keskiarvo", "Mediaani", "\\sigma")
rownames(times.descr) <- control.rownames
times.descr
```