NEWS Director

kobayashikorio@gmail.com
2017/10/23

NEWS Director automatically search North and shows novles relating the bearings to remember your old time of reading. The project is intended to evaluate the handling of multiple modules and managing of moving parts.

Software and hardware challenges
1. mechanical display rotation
2. manipulate a stepping mortor
3. sensing of earth magnetic field
4. monitoring posture condition with accelerometer
5. manipulation for multiple I2C devices
6. construction system package on the RaspberryPi Zero case

hardware connection
1. RaspberryPi to stepping mortor driver

pin:BCM	-> device
1 :5V	-> Vcc
6 :GND	-> GND
7:BCM04	-> IN4
11:BCM17	-> IN3
13:BCM27	-> IN2
15:BCM22	-> IN1

2. RaspberryPi to magnetometer-accelerometer and SSD1306

3:BCM02	-> SDA(yellow-green)
5:BCM03	-> SCL (orange-white)
1:3.3V	-> Vcc (red-red)
9:GND	-> GND (blown-black)

3. shutdown tact switch

31:BCM06	-> switch
32:BCM12	-> switch

Standalone setting
To start the program automatically after the booting process, all code should be written in text and add a following line at the begnning,

#! /opt/Wolfram/WolframEngine/10.3/Executables/math -script

then put this file to directory /home/pi, as the name “storyteller”, then chmod +x the script file. When code maintenace is required, set pin7 to GND and the booting process to escape shut-down.

* to autostart the script, you may add next line into the cron table with “$ crontab -e”,
@reboot /home/pi/storyteller >>/temp/log 2>>/temp/logerr
* to start cron, you may edit /etc/rc.local and insert next line before “exit 0”,
/etc/init.d/cron start

- **NEWS-director**

```
(* get data *)
Get["/home/pi/Desktop/fontTable.nb"];
Get["/home/pi/Desktop/bearingText.nb"];
```

```
(* setup SSD1306 display *)
chipAdr = "0x3c";
cmdStr := {"/usr/sbin/i2cset -y 1", chipAdr, "0x00"};
dataStr := {"/usr/sbin/i2cset -y 1", chipAdr, "0x40"};
rPut[r_] := WriteLine[process, StringRiffle[Join[cmdStr, {r}]]];
setCursor[r_] := (
    rPut[StringJoin["0xb", ToString[r]]];
    rPut["0x10"]; rPut["0x00"]
    (* lower nible of column to 2 for 1.3, 0 for .95inch display *)
  );
clrLine[r_] := (
    blnkStr = Join[dataStr, ConstantArray["0x00", 32], {"i"}];
    Pause[0.1];
    setCursor[r];
    Table[WriteLine[process, StringRiffle[blnkStr]], {4}]
  );
clrScreen := Table[clrLine[i], {i, 7, 0, -1}];
dispLine[r_, str_] := (
    clrLine[r];
    setCursor[r];
    cstr = DeleteMissing[Map[fontTable, Characters[str]]];
    dcom = Map[StringRiffle[Join[dataStr, #, {"i"}]] &, cstr];
    Map[WriteLine[process, #] &, dcom];
  );
process = StartProcess[$SystemShell];
clrScreen;
rPut["0x8d"]; (* 8d: charge pump on*)
rPut["0x14"]; (* 14: enable charge pump*)
rPut["0xaf"]; (* af: display on in normal mode*)
rPut["0xa1"];(*  a1: set segment remap to reverse*)
dispLine[6, "display ready"];


(* setup GPIO for stepping mortor *)
dispLine[5, "mortor driver preparing ..."];
DeviceConfigure["GPIO", {4 → "Output", 17 → "Output", 27 → "Output", 22 → "Output"}];
cw := (DeviceWrite["GPIO", {4 → 0, 22 → 1}];
    DeviceWrite["GPIO", {22 → 0, 27 → 1}];
    DeviceWrite["GPIO", {27 → 0, 17 → 1}];
    DeviceWrite["GPIO", {17 → 0, 4 → 1}]);
ccw := (DeviceWrite["GPIO", {22 → 0, 4 → 1}];
    DeviceWrite["GPIO", {4 → 0, 17 → 1}];
    DeviceWrite["GPIO", {17 → 0, 27 → 1}];
    DeviceWrite["GPIO", {27 → 0, 22 → 1}]);
stop := DeviceWrite["GPIO", {4 → 0, 17 → 0, 27 → 0, 22 → 0}];


(* setup I2C interface *)
rGet[cad_, r_] :=
  (WriteLine[process, StringRiffle[{"/usr/sbin/i2cget -y 1", cad, r}]];
    ReadLine[process]);
rPut[cad_, r_, d_] :=
  WriteLine[process, StringRiffle[{"/usr/sbin/i2cset -y 1", cad, r, d}]];
(* get signed short integer *)
ssGet[cad_, {reg1_, reg2_}] := (r = Flatten[{rGet[cad, reg1], rGet[cad, reg2]}];
    r1 = Map[FromDigits[#, 16] &, StringReplace[r, "0x" → ""]];
    r2 = BitShiftLeft[r1[[1]], 8] + r1[[2]];
    If[r2 > 32767, -32768 + (r2 - 32768), r2]);


 (* setup magnetometer & accelerometer *)
dispLine[4, "magneto-meter preparing ..."];
mChipAdr = "0x1e";
mreg = Association[craReg → "0x00", mrReg → "0x02", outXM → {"0x03", "0x04"},
    outYM → {"0x07", "0x08"}, outZM → {"0x05", "0x06"}, tmp → {"0x31", "0x32"}
];
(* TEMP enable and data rate 30Hz *)
```

```
rPut[mChipAdr, mreg@craReg, "0x94"];
(* continuous conversion *)
rPut[mChipAdr, mreg@mrReg, "0x00"];
 (* setup accerometer *)
aChipAdr = "0x19";
areg = Association[cntlReg1 → "0x20", mrReg → "0x02",
   outXA → {"0x29", "0x28"}, outYA → {"0x2b", "0x2a"}, outZA → {"0x2d", "0x2c"}];
(* set output data rate to 50Hz*)
rPut[aChipAdr, areg@cntlReg1, "0x47"];


(* check the posture of the system with accelerometer *)
While[Norm[{ssGet[aChipAdr, areg@outXA], ssGet[aChipAdr, areg@outYA]}] > 2000.,
 dispLine[3, "put me on flat place"];
 Pause[2.0]]


(* lets rotate base 360 degree to get earth magnet field data *)
dispLine[3, "sencing earth magnet field ..."];
dc = Reap[Do[cw;
   (* get magnet data once per 8 motor step *)
   If[Mod[i, 8, 1] == 1,
    Sow[{ssGet[mChipAdr, mreg@tmp] / 16., ssGet[mChipAdr, mreg@outXM],
      ssGet[mChipAdr, mreg@outYM], ssGet[mChipAdr, mreg@outZM]}]];
   , {i, 1, 64 * 8}]]; stop;
Pause[0.1];
Do[ccw, {i, 1, 64 * 8}];
stop;


(* find the magnet N direction and adjust the system offset *)
dispLine[2, "sytem ready to start ..."];
(* adjust-ring adjusts angle between magnetometer x-direction and display face *)
adjustRing = 16;
d = (dc /. Null → Nothing)[[1, 1]];
(* fitting Sine curve to observation, detect North position *)
f = Transpose[{Range[2 Pi / 64., 2 Pi, 2 Pi / 64.], d[[All, 2]]}];
coef = FindFit[f, a + b * Sin[c + x], {a, b, c}, x];
(* next program line shows the result of fitting *)
(*ListLinePlot[{d[[All,2]],Table[a+b*Sin[c+x]/.coef,{x,2Pi/64.,2Pi,2Pi/64.}]}]*)
bearingName = {"N", "NE", "E", "SE", "S", "SW", "W", "NW"};
(* from the pahse of fitting result a max value can be estimated *)
(* in some case, find fit returns a minus amplitude value *)
{ss, phai} = {b > 0, c} /. coef;
Npos = Round[Mod[-phai + If[ss, 5 Pi / 2, 7 Pi / 2], 2 Pi] / (2 Pi) * 64];
Npos = Mod[Npos + adjustRing, 64, 1];



(* management module for rotating magnetometer and display *)
Module[{ringPos, nextPos},
  set[ring[npos_]] := (
    posList = Union[Flatten[Join[{Range[npos, 64, 8], Range[npos, 1, -8]}]]];
posListR = RotateLeft[posList, Position[posList, npos][[1, 1]] - 1];
bearing = Association[Map[Apply[Rule, #] &, Transpose[{bearingName, posListR}]]];
    ringPos = 1; Return[bearing];
   );
  get[ring[direc_]] := (
    nextPos = bearing[direc];
    If[nextPos == ringPos, rcode = {Hold[stop], 0}];
    If[nextPos > ringPos, rcode = {Hold[cw], nextPos - ringPos}];
    If[nextPos < ringPos, rcode = {Hold[ccw], ringPos - nextPos}];
    ringPos = nextPos;
    Return[rcode];
   );
  rewind[ring] := (
    If[ringPos == 1,
     rcode = {Hold[stop], 0},
     rcode = {Hold[ccw], ringPos - 1}];
```

```
    ringPos = 1;
    Return[rcode];
   );
 ];
(* stepping motor handler*)
winder[dir_, n_] := Do[ReleaseHold[dir], {i, n * 8}];
(* set shutdown switch *)
WriteLine[process, "/usr/bin/gpio -g mode 12 in"];
(* initialize adjust ring *)
set[ring[Npos]];


 (* main loop *)
While[True,
  (* system shut-down process routine *)
  WriteLine[process, "/usr/bin/gpio -g read 12"];
  If[ReadLine[process] == "1",
   clrScreen;
   dispLine[2, "shutdown process"];
   Apply[winder, rewind[ring]]; stop;
   Exit[];
   clrScreen;
  ];
  nextBearing = RandomChoice[bearingName];
(* choice of item *)
  ctxt = RandomChoice[bearingTxt[nextBearing]];
splist = StringSplit[ctxt, ","];
clrScreen; stline = 5;
  Do[dispLine[stline--, splist[[i]]], {i, 1, Length[splist]}];
  (* go to the direction *)
  Apply[winder, get[ring[nextBearing]]]; stop;
  Pause[1.0];

 ];

(* if the equipent movement was aborted, adjust the position by the next command *)
Apply[winder, rewind[ring]]; stop;
```