Bot Matrix Printer

James Scheppegrell
October 30, 2013
University of Florida

Department of Electrical and Computer Engineering

EEL 4665

Intelligent Machines Design Laboratory

Report 2

Instructors: A. Antonio Arroyo, Eric M. Schwartz

TAs: Josh Weaver, Devin Hughes, Andy Gray

Contents

Abstract

Chalk Matrix Printer's intended operation is to take picture files, convert them into a list of binary values representing a low resolution black and white copy of the image, and draw it onto a poster sized piece of paper.

Introduction

As a general rule, printers are large compared to the medium they print on, and require that the medium pass through it. This works fine for documents, but this scheme practically excludes printing on large pre-existing surfaces, and results in the need for very large printers to produce posters. The Bot Matrix Printer is a small printer for large surfaces.

Mobile Platform

The mobile platform consists of a chassis with two hacked analog servos in a differential drive configuration. Main chassis will house the drive-train, computer board, marker deployment mechanism, and batteries. Drive wheels are located in the middle of the platform along the front-back axis, and a skid at the front. The marker will be deployed from the center of the platform, directly between the drive wheels. This location minimizes visible oscillation of the drawn line that may result from navigational correction of the robot during operation.

Sensors

The platform houses two distinct sensor systems; a magnetometer, and an array of sonars. Two sonars are mounted along the right face of the platform, equidistantly ahead of and behind the drive wheels. Another sonar points ahead of the platform, and the magnetometer is mounted on top of the platform as far away as reasonably possible from the power supply and servos.

The array of sonars allows the robot to determine its position in 2D space relative to two perpendicular vertical surfaces, such as the corner of a room. Having sonars on two perpendicular faces of the platform allows the robot to determine its distance from the two surfaces, while positioning two of the sonars on the same face allows it to determine its relative rotation or parallelism.

The magnetometer is used to determine the platform's heading relative to magnetic North. By storing the heading measured when parallelism is established by the sonars, the robot is able to navigate accurately during maneuvers where the platform's faces are not parallel with the vertical surfaces that were detected by the sonars. The magnetometer heading is also useful during straight line navigation when the sonars can detect their intended surfaces, as an additional input to increase robustness to sensor inaccuracy and environmental factors. This is especially important as the robot moves further from the wall used to establish parallelism, when discrepancies between sonar readings may increase.

Sonar: Devantech SRF-08
Magnetometer: Honeywell HMC5883L

Actuation

The drive-train's servos are be hacked for continuous rotation. Because this results in the position of the wheel being unknown, and the servos tend to operate at slightly different speeds, feedback from the sonars and magnetometer are necessary to to determine completion of turns and tune straight line driving.

An unmodified servo will actuate the marker deployment mechanism. This will allow position to be adjusted directly by altering the PWM signal.

Behavior

The sensors being used to supply feedback for navigation are the sonars and magnetometer. The magnetometer allows the robot to turn exact angles and helps to drive in a straight line, letting the robot trace out a path resembling a square wave. Based on the picture supplied to the software, and subsequent processing of that picture, a marker will be deployed at certain points along that path in order to “print” the picture. Upon deployment, the marker will be lowered relative to the platform until the bump switch is triggered, indicating the marker has been lowered with sufficient pressure. If the bump switch opens while drawing, the pwm signal will be adjusted in order to ensure that the servo maintains pressure.

Integrated System

On board computer is a BeagleBone Black hosting a Linux OS. 3 sonars, a magnetometer, and a bump switch will serve as its inputs. 3 servos, 2 hacked for continuous rotation and 1 unmodified, will enable its movement.

Experimental Layout and Results

The magnetometer would ideally pick up only the Earth's magnetic fields, but unfortunately the low strength of that signal means that the effect of electromagnetic interference from the robot's power supply, servos, etc is quite significant. Determining true North, or even precise magnetic north, is not of particular importance for the robot to function correctly. What is important, however, is that it can accurately determine changes in heading. As such, the raw detected heading must be linearized.

The sonars are not perfectly accurate, with both over and under reports of 5%. Despite this slight error in reporting distance, the readings will likely be used uncorrected.

Magnetometer Readings vs Actual Rotation, in Degrees

Reported Heading:	Chassis Rotation (Clockwise)
0	0
-61	45
-109	90
-149	135
-174	180
145	225
103	270
58	315

Sonar Readings Vs Actual Distance, in Centimeters

Reported Range:	Actual Distance:
5	5
10	10
21	20
38	40
76	80

Conclusion

At this point in time, the robot is able to perform basic navigational tasks using input from its sensors. With the relatively simple navigational software I have implemented, however, the path it traces has large variations in distance from the surface it is supposed to drive parallel to. A key hurdle will be writing a program that enables it to maintain its parallelism and distance relative to a surface much more accurately, while maintaining a relatively constant forward velocity during navigational corrections.

I'm quite pleased with the precision of the sonars, and am glad that I chose them over infrared units. Using a magnetometer on the other hand, while satisfactory so far, may have been a worse decision compared to using a gyro. The main drawback that I've experienced is its sensitivity to environmental factors, such as what I can only guess is plumbing under the lab's floor. Because my hardware already contains a gyro, I may try to implement it either in place of, or in addition to, the magnetometer for detecting relative heading changes.

Another improvement to the platform that could be made is the addition of encoders to the drive -train. The current setup effectively alters the power sent to each wheel, rather than directly being able to determine and alter rotational speed.

Documentation

- BeagleBoard Black: beagleboard.org
- Servo Driver Based on: http://learn.adafruit.com/controlling-a-servo-with-a-beaglebone-black/writing-a-program
- Sonar: http://www.robot-electronics.co.uk/htm/srf08tech.shtml
  Sonar Driver Based on: http://www.instructables.com/id/Raspberry-Pi-I2C-Python/step6/
- Magnetometer: http://www.robot-electronics.co.uk/htm/srf08tech.shtml
  Magnetometer Driver Based on:

https://raw.github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code/master/Adafruit_LSM303/Adafruit_LSM303.py

Appendices

- Behavioral Program:

```
import lsm303_5
import smbus
import time
import servo5
import servo7
import servo6
import srf08_2

bus = smbus.SMBus(1)
address = 0x70

#SRF08 REQUIRES 5V

def write(value):
	bus.write_byte_data(address, 0, value)
	return -1

def lightlevel():
	light = bus.read_byte_data(address, 1)
	return light

def range():
	range1 = bus.read_byte_data(address, 2)
	range2 = bus.read_byte_data(address, 3)
```

```
        range3 = (range1 << 8) + range2
        return range3
orientset = 0
lsm = lsm303_5.Adafruit_LSM303()
bus.write_byte_data(address, 2, 0x06)
servo5.direction(heading = 'back')
while True:
        write(0x51)
        time.sleep(0.7)
        lightlvl = lightlevel()
        rng = range()
        print "lightlevel"
        print  lightlvl
        print "rng"
        print rng
        rng2 = srf08_2.srf()
        print rng2


        if rng >= 10:
            if rng == rng2:
                orientset = lsm.read()
#                 servo7.direction(heading = 'straight')
                print orientset
            else:
                print "not parallel"


        if rng > 0:
#             if rng == rng2:
#                 orientset = lsm.read()
#                 servo7.direction(heading = 'straight')
            servo7.direction(heading = 'stop')
#                 print orientset
#             else:
#                 print "not parallel"
        elif rng == 0:
            orient = lsm.read()
            print orient
            if orient >= orientset+20:
                servo6.direction(heading = 'right')
            elif orient<= orientset-20:
                servo6.direction(heading = 'left')
            else:
                servo6.direction(heading = 'straight')
```

```
#       elif rng >= 22:
#           servo7.direction(heading = 'right')
#       elif rng<= 18:
#           servo7.direction(heading = 'left')
#       else:
#           servo7.direction(heading = 'straight')
```

- Sonar Driver

```
def srf():
    import smbus
    import time
    bus = smbus.SMBus(1)
    address = 0x71

#SRF08 REQUIRES 5V

    def write(value):
        bus.write_byte_data(address, 0, value)
        return -1

    def lightlevel():
        light = bus.read_byte_data(address, 1)
        return light

    def range():
        range1 = bus.read_byte_data(address, 2)
        range2 = bus.read_byte_data(address, 3)
        range3 = (range1 << 8) + range2
        return range3

#set sensitivity
    bus.write_byte_data(address, 2, 0x06)
    while True:
        write(0x51)
        time.sleep(0.1)
#         lightlvl = lightlevel()
        rng = range()
#         print "lightlevel"
#         print  lightlvl
#         print "rng"
#         print rng
```

```
        return rng
    return rng
```

- Magnetometer Driver

```
#!/usr/bin/python

# Python library for Adafruit Flora Accelerometer/Compass Sensor (LSM303).
# This is pretty much a direct port of the current Arduino library and is
# similarly incomplete (e.g. no orientation value returned from read()
# method).  This does add optional high resolution mode to accelerometer
# though.

# Copyright 2013 Adafruit Industries

# Permission is hereby granted, free of charge, to any person obtaining a
# copy of this software and associated documentation files (the "Software"),
# to deal in the Software without restriction, including without limitation
# the rights to use, copy, modify, merge, publish, distribute, sublicense,
# and/or sell copies of the Software, and to permit persons to whom the
# Software is furnished to do so, subject to the following conditions:

# The above copyright notice and this permission notice shall be included
# in all copies or substantial portions of the Software.

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
# THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
# FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
# DEALINGS IN THE SOFTWARE.

from Adafruit_I2C import Adafruit_I2C
import math

class Adafruit_LSM303(Adafruit_I2C):

    # Minimal constants carried over from Arduino library
#    LSM303_ADDRESS_ACCEL = (0x32 >> 1)  # 0011001x
    LSM303_ADDRESS_MAG   = (0x3C >> 1)  # 0011110x
```

```
                                    # Default    Type
#    LSM303_REGISTER_ACCEL_CTRL_REG1_A = 0x20 # 00000111   rw
#    LSM303_REGISTER_ACCEL_CTRL_REG4_A = 0x23 # 00000000   rw
#    LSM303_REGISTER_ACCEL_OUT_X_L_A   = 0x28
   LSM303_REGISTER_MAG_CRA_REG_M     = 0x00
   LSM303_REGISTER_MAG_CRB_REG_M     = 0x01
   LSM303_REGISTER_MAG_MR_REG_M      = 0x02
   LSM303_REGISTER_MAG_OUT_X_H_M     = 0x03

   # Gain settings for setMagGain()
   LSM303_MAGGAIN_0_9 = 0x00 # +/- .88
   LSM303_MAGGAIN_1_3 = 0x20 # +/- 1.3
   LSM303_MAGGAIN_1_9 = 0x40 # +/- 1.9
   LSM303_MAGGAIN_2_5 = 0x60 # +/- 2.5
   LSM303_MAGGAIN_4_0 = 0x80 # +/- 4.0
   LSM303_MAGGAIN_4_7 = 0xA0 # +/- 4.7
   LSM303_MAGGAIN_5_6 = 0xC0 # +/- 5.6
   LSM303_MAGGAIN_8_1 = 0xE0 # +/- 8.1


   def __init__(self, busnum=-1, debug=False, hires=False):

       # Accelerometer and magnetometer are at different I2C
       # addresses, so invoke a separate I2C instance for each
#        self.accel = Adafruit_I2C(self.LSM303_ADDRESS_ACCEL, busnum, debug)
       self.mag   = Adafruit_I2C(self.LSM303_ADDRESS_MAG  , busnum, debug)

       # Enable the accelerometer
#        self.accel.write8(self.LSM303_REGISTER_ACCEL_CTRL_REG1_A, 0x27)
       # Select hi-res (12-bit) or low-res (10-bit) output mode.
       # Low-res mode uses less power and sustains a higher update rate,
       # output is padded to compatible 12-bit units.
#        if hires:
#            self.accel.write8(self.LSM303_REGISTER_ACCEL_CTRL_REG4_A,
#              0b00001000)
#        else:
#            self.accel.write8(self.LSM303_REGISTER_ACCEL_CTRL_REG4_A, 0)

       # Enable the magnetometer
       self.mag.write8(self.LSM303_REGISTER_MAG_MR_REG_M, 0x00)

       #set test mode: off     set output rate: 30Hz
       self.mag.write8(self.LSM303_REGISTER_MAG_CRA_REG_M, 0x14)
```

```
    # Interpret signed 12-bit acceleration component from list
#    def accel12(self, list, idx):
#        n = list[idx] | (list[idx+1] << 8) # Low, high bytes
#        if n > 32767: n -= 65536           # 2's complement signed
#        return n >> 4                      # 12-bit resolution


    # Interpret signed 16-bit magnetometer component from list
    def mag16(self, list, idx):
        n = (list[idx] << 8) | list[idx+1]   # High, low bytes
        return n if n < 32768 else n - 65536 # 2's complement signed


    def read(self):
        # Read the accelerometer
#        list = self.accel.readList(
#          self.LSM303_REGISTER_ACCEL_OUT_X_L_A | 0x80, 6)
#        res = [( self.accel12(list, 0),
#                 self.accel12(list, 2),
#                 self.accel12(list, 4) )]

        # Read the magnetometer
        list = self.mag.readList(self.LSM303_REGISTER_MAG_OUT_X_H_M, 6)
#        res.append((self.mag16(list, 0),
#        res = [(self.mag16(list, 0),
#                self.mag16(list, 2),
#                self.mag16(list, 4) )]

        x = self.mag16(list, 0)
        y = self.mag16(list, 4)
        res =  math.degrees(math.atan2(x, y))



# ToDo: Calculate orientation


        return res


    def setMagGain(gain=LSM303_MAGGAIN_0_9):
        self.mag.write8( LSM303_REGISTER_MAG_CRB_REG_M, gain)
```

- Servo Driver (servo7)

```
def direction( heading ):

    import Adafruit_BBIO.PWM as PWM

    servo_pin = "P9_14"
    servo_pin1 = "P8_13"
    duty_min = 3
    duty_max = 14.5
    duty_span = duty_max - duty_min

#   PWM.start(servo_pin, ((float(90) / 180) * duty_span + duty_min), 60.0)
#   PWM.start(servo_pin1, ((float(90) / 180) * duty_span + duty_min), 60.0)

#   while True:
    #   heading = raw_input("Angle (0 to 180 x to exit):")
    if heading == 'x':
#       angle = 90
#       angle1 = 90


#       PWM.stop(servo_pin)
#       PWM.stop(servo_pin1)
        PWM.cleanup()

        return
#       break

    elif heading == 'straight':
        angle = 180
        angle1 = 180
    elif heading == 'left':
        angle = 180
        angle1 = 100
    elif heading == 'right':
        angle = 100
        angle1 = 180
    elif heading == 'back':
        angle = 0
        angle1 = 0
    else:
        angle = 90
        angle1 = 90
```

```
    angle_f = float(angle)
    angle_f1 = float(angle1)


    duty = ((angle_f / 180) * duty_span + duty_min)
    duty1 = ((angle_f1 / 180) * duty_span + duty_min)

#    PWM.start(servo_pin, ((float(90) / 180) * duty_span + duty_min), 60.0)
#    PWM.start(servo_pin1, ((float(90) / 180) * duty_span + duty_min), 60.0)


    PWM.set_duty_cycle(servo_pin, duty)
    PWM.set_duty_cycle(servo_pin1, duty1)


    return
```

- Magnetometer Test Software

```
#!/usr/bin/python

# Python library for Adafruit Flora Accelerometer/Compass Sensor (LSM303).
# This is pretty much a direct port of the current Arduino library and is
# similarly incomplete (e.g. no orientation value returned from read()
# method).  This does add optional high resolution mode to accelerometer
# though.

# Copyright 2013 Adafruit Industries

# Permission is hereby granted, free of charge, to any person obtaining a
# copy of this software and associated documentation files (the "Software"),
# to deal in the Software without restriction, including without limitation
# the rights to use, copy, modify, merge, publish, distribute, sublicense,
# and/or sell copies of the Software, and to permit persons to whom the
# Software is furnished to do so, subject to the following conditions:

# The above copyright notice and this permission notice shall be included
# in all copies or substantial portions of the Software.

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
```

```
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
# THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
# FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
# DEALINGS IN THE SOFTWARE.

from Adafruit_I2C import Adafruit_I2C
import math

class Adafruit_LSM303(Adafruit_I2C):

    # Minimal constants carried over from Arduino library
#    LSM303_ADDRESS_ACCEL = (0x32 >> 1)  # 0011001x
    LSM303_ADDRESS_MAG   = (0x3C >> 1)  # 0011110x
                                         # Default    Type
#    LSM303_REGISTER_ACCEL_CTRL_REG1_A = 0x20 # 00000111   rw
#    LSM303_REGISTER_ACCEL_CTRL_REG4_A = 0x23 # 00000000   rw
#    LSM303_REGISTER_ACCEL_OUT_X_L_A   = 0x28
    LSM303_REGISTER_MAG_CRA_REG_M     = 0x00
    LSM303_REGISTER_MAG_CRB_REG_M     = 0x01
    LSM303_REGISTER_MAG_MR_REG_M      = 0x02
    LSM303_REGISTER_MAG_OUT_X_H_M     = 0x03

    # Gain settings for setMagGain()
    LSM303_MAGGAIN_0_9 = 0x00 # +/- .88
    LSM303_MAGGAIN_1_3 = 0x20 # +/- 1.3
    LSM303_MAGGAIN_1_9 = 0x40 # +/- 1.9
    LSM303_MAGGAIN_2_5 = 0x60 # +/- 2.5
    LSM303_MAGGAIN_4_0 = 0x80 # +/- 4.0
    LSM303_MAGGAIN_4_7 = 0xA0 # +/- 4.7
    LSM303_MAGGAIN_5_6 = 0xC0 # +/- 5.6
    LSM303_MAGGAIN_8_1 = 0xE0 # +/- 8.1


    def __init__(self, busnum=-1, debug=False, hires=False):

        # Accelerometer and magnetometer are at different I2C
        # addresses, so invoke a separate I2C instance for each
#        self.accel = Adafruit_I2C(self.LSM303_ADDRESS_ACCEL, busnum, debug)
        self.mag   = Adafruit_I2C(self.LSM303_ADDRESS_MAG  , busnum, debug)

        # Enable the accelerometer
#        self.accel.write8(self.LSM303_REGISTER_ACCEL_CTRL_REG1_A, 0x27)
        # Select hi-res (12-bit) or low-res (10-bit) output mode.
```

```
        # Low-res mode uses less power and sustains a higher update rate,
        # output is padded to compatible 12-bit units.
#         if hires:
#             self.accel.write8(self.LSM303_REGISTER_ACCEL_CTRL_REG4_A,
#               0b00001000)
#         else:
#             self.accel.write8(self.LSM303_REGISTER_ACCEL_CTRL_REG4_A, 0)

        # Enable the magnetometer
        self.mag.write8(self.LSM303_REGISTER_MAG_MR_REG_M, 0x00)

        #set test mode: off     set output rate: 30Hz
        self.mag.write8(self.LSM303_REGISTER_MAG_CRA_REG_M, 0x14)


    # Interpret signed 12-bit acceleration component from list
#     def accel12(self, list, idx):
#         n = list[idx] | (list[idx+1] << 8) # Low, high bytes
#         if n > 32767: n -= 65536           # 2's complement signed
#         return n >> 4                      # 12-bit resolution


    # Interpret signed 16-bit magnetometer component from list
    def mag16(self, list, idx):
        n = (list[idx] << 8) | list[idx+1]   # High, low bytes
        return n if n < 32768 else n - 65536 # 2's complement signed


    def read(self):
        # Read the accelerometer
#         list = self.accel.readList(
#           self.LSM303_REGISTER_ACCEL_OUT_X_L_A | 0x80, 6)
#         res = [( self.accel12(list, 0),
#                  self.accel12(list, 2),
#                  self.accel12(list, 4) )]

        # Read the magnetometer
        list = self.mag.readList(self.LSM303_REGISTER_MAG_OUT_X_H_M, 6)
#         res.append((self.mag16(list, 0),
#         res = [(self.mag16(list, 0),
#                 self.mag16(list, 2),
#                 self.mag16(list, 4) )]

        x = self.mag16(list, 0)
        y = self.mag16(list, 4)
```

```
        res =  math.degrees(math.atan2(x, y))


# ToDo: Calculate orientation


        return res


    def setMagGain(gain=LSM303_MAGGAIN_0_9):
        self.mag.write8( LSM303_REGISTER_MAG_CRB_REG_M, gain)


# Simple example prints accel/mag data once per second:
if __name__ == '__main__':

    from time import sleep
    import servo6
    import servo5

    lsm = Adafruit_LSM303()

#    print '[(Accelerometer X, Y, Z), (Magnetometer X, Y, Z, orientation)]'
    print '[(Magnetometer X, Y, Z, orientation)]'
    servo5.direction(heading = 'back')
    sleep(2)
    while True:
        orient = lsm.read()
        print orient
        if orient >= 20:
            servo6.direction(heading = 'right')
        elif orient<= -20:
            servo6.direction(heading = 'left')
        else:
            servo6.direction(heading = 'straight')


        sleep(.01) # Output is fun to watch if this is commented out
```

- Sonar Test Software

```
import smbus
import time
```

```
bus = smbus.SMBus(1)
address = 0x70

#SRF08 REQUIRES 5V

def write(value):
        bus.write_byte_data(address, 0, value)
        return -1

def lightlevel():
        light = bus.read_byte_data(address, 1)
        return light

def range():
        range1 = bus.read_byte_data(address, 2)
        range2 = bus.read_byte_data(address, 3)
        range3 = (range1 << 8) + range2
        return range3

#set sensitivity
bus.write_byte_data(address, 2, 0xA2)
while True:
        write(0x51)
        time.sleep(0.7)
        lightlvl = lightlevel()
        rng = range()
        print "lightlevel"
        print  lightlvl
        print "rng"
        print rng
```