Daniel Diamont

EE 445L

4/6/18

LAB 8 REPORT

OBJECTIVES

Write low-level software drivers for the final project.

Requirements Document: Lab 8

1. Overview

1.1. Objectives: Why are we doing this project? What is the purpose?

The objectives of this project are to design, build, and test a useful embedded system. Educationally, students are learning about designing Printed Circuit Boards, interfacing hardware and software for a functional embedded system, and mechanical enclosures for an embedded system. My goal is to create an autonomous vehicle that uses computer vision and a discrete PID control algorithm to guide itself through an obstacle course.

1.2. Roles and Responsibilities: Who will do what? Who are the clients?

EE445L students are the engineers and the TA, Professor Valvano, and the competition judges are the clients. All tasks will be performed by Daniel Diamont.

- Tasks:
  - Requirements Document
  - PCB Schematic
  - Bill of Materials
  - PCB Design
  - Drivers for hardware interfacing
  - PID controller software
  - Computer-Vision software
  - Enclosure Design
  - Fully functional embedded system

1.4. Interactions with Existing Systems: Include this if you are connecting to another board

The system will use a Raspberry Pi 2 Model B board to perform the video-processing, PID control, and DC motor control. The camera will be connected directly to the Pi's Camera Interface module. Additionally, the system will use a printed circuit board – designed by the student – to interface the Raspberry Pi with DC motors, and an accelerometer + gyro. The system will be powered by a +9.0V battery, and a Linear Dropout Regulator will be used to step down the voltage to +5V for the Pi and the peripherals.

2. Function Description

2.1. Functionality: What will the system do precisely?

The robot car will interface over Wi-Fi to a Python application on a PC, which will either send a command enabling full autonomy, or manually override directional control of the DC motors for demonstration purposes. Fully-autonomous mode will allow the robot car to capture video of its immediate forward area, process the video to recognize markings to follow and obstacles to navigate, and plot and execute a path through the course using a software PID controller.

The robot car's software will use multithreading, parallel programming, and interrupt-driven concurrent programming techniques to handle Wi-Fi communication, video-processing, PID control, motor control, and sensor data acquisition at the same time.

2.4. Performance: Define the measures and describe how they will be determined.

The system will be judged qualitatively by 9 measures. First, the software modules must be easy to understand and well-organized. Second, the system must satisfy the two input/two output requirements and the multiple ISR requirement. Third, the turnover rate between frame capture and motor control output must be real-time. Fourth, the system must be able to reliably communicate from the controller PC with minimal TCP packet loss. Fifth, the time to run one execution of all threads except for the main thread must be minimized to reduce overhead in the real-time system. Sixth, the design will be judged on whether the robot-car and embedded system enclosure are well-thought out and functional. Seventh, the PCB design must be simple and elegant. Eight, there must be a thorough Bill of Materials to calculate the final cost of the project. Ninth, the system must have mechanisms in place to make it simple to debug the hardware and software in a modular fashion.

2.5. Usability: Describe the interfaces. Be quantitative if possible.

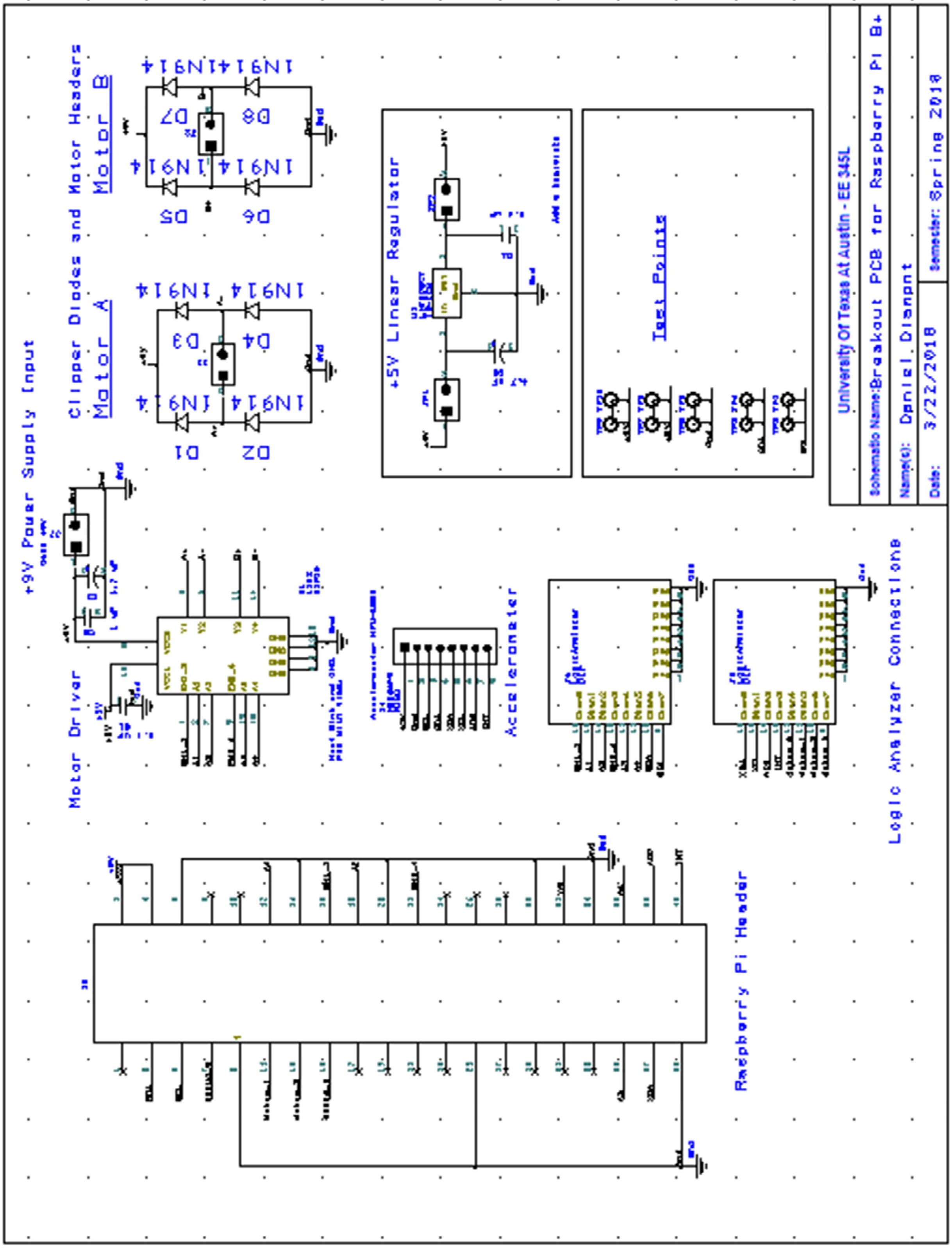
I will use a Raspberry Pi 2 Model B. The Pi will be interfaced with an 5 megapixel ArduCAM OV5647 camera module for the video-capture, an 802.11(b/g/n) Wi-Fi USB adapter for communication, the Raspbian Operating System, Python 2.7 libraries like RPi.GPIO and smbus for low-level drivers, OpenCV2 libraries for computer vision, PID, and motor control, L293 IC for the motor driver, and MPU 6050 for the gyro + accelerometer.

3. Deliverables

3.1. Reports: Reports for Labs 7, 8, and 11 will be written

HARDWARE DESIGN

- See attached UTX-2018S102.sch file.



SOFTWARE DESIGN

- See attached *.py files in /drivers/

Video Driver

```python
# Import the necessary python libraries

from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2

#Initialize camera and grab a reference to the raw camera capture

res_width = 480
res_length = 320
fr = 30

camera = PiCamera()
camera.resolution = (res_width, res_length)
camera.framerate = fr
rawCapture = PiRGBArray(camera, size=(res_width, res_length))

#allow camera to warm-up
time.sleep(0.1)

# This code sets up a video feed from your system's default web cam
#cap = cv2.VideoCapture(0)

# Capture rames from the camera
for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
    # Read a frame from the camera
    image = frame.array

    # Display the frame in a window
    #cv2.imshow('frame', frame)
    cv2.imshow('image', image)

    rawCapture.truncate(0)

    # This code will quit the program if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release system webcam and close all windows
cv2.destroyAllWindows()

time.sleep(1)
```

Motor Controller Driver

```python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)

#make sure to set the correct GPIO pins according to SCH
Motor1A = 12
Motor1B = 18
Motor1E = 16

Motor2A = 23
Motor2B = 21 #change later
Motor2E = 19

#set up GPIO
def gpio_setup():
    #setup motor one
    GPIO.setup(Motor1A,GPIO.OUT)
    GPIO.setup(Motor1B,GPIO.OUT)
    GPIO.setup(Motor1E,GPIO.OUT)
    #setup motor two
    GPIO.setup(Motor2A,GPIO.OUT)
    GPIO.setup(Motor2B,GPIO.OUT)
    GPIO.setup(Motor2E,GPIO.OUT)

def move_forwards():
    #left motor forwards
    GPIO.output(Motor1A,GPIO.HIGH)
    GPIO.output(Motor1B,GPIO.LOW)
    GPIO.output(Motor1E,GPIO.HIGH)
    #right motor forwards
    GPIO.output(Motor2A,GPIO.HIGH)
    GPIO.output(Motor2B,GPIO.LOW)
    GPIO.output(Motor2E,GPIO.HIGH)

def move_right():
    #left motor forwards
    GPIO.output(Motor1A,GPIO.HIGH)
    GPIO.output(Motor1B,GPIO.LOW)
    GPIO.output(Motor1E,GPIO.HIGH)
    #right motor stop
    GPIO.output(Motor2A,GPIO.LOW)
    GPIO.output(Motor2B,GPIO.LOW)
    GPIO.output(Motor2E,GPIO.LOW)
```

```python
        GPIO.output(Motor1E,GPIO.HIGH)
        #right motor stop
        GPIO.output(Motor2A,GPIO.LOW)
        GPIO.output(Motor2B,GPIO.LOW)
        GPIO.output(Motor2E,GPIO.LOW)

    def move_left():
        #left motor stop
        GPIO.output(Motor1A,GPIO.LOW)
        GPIO.output(Motor1B,GPIO.LOW)
        GPIO.output(Motor1E,GPIO.LOW)
        #right motor high
        GPIO.output(Motor2A,GPIO.HIGH)
        GPIO.output(Motor2B,GPIO.LOW)
        GPIO.output(Motor2E,GPIO.HIGH)

    def move_backwards():
        GPIO.output(Motor1A,GPIO.LOW)
        GPIO.output(Motor1B,GPIO.HIGH)
        GPIO.output(Motor1E,GPIO.HIGH)

        GPIO.output(Motor2A,GPIO.LOW)
        GPIO.output(Motor2B,GPIO.HIGH)
        GPIO.output(Motor2E,GPIO.HIGH)

    def stop_motors():
        GPIO.output(Motor1E,GPIO.LOW)
        GPIO.output(Motor2E,GPIO.LOW)

    gpio_setup()

    while True:
        move_forwards()
        print('forward')
        time.sleep(1)
        move_backwards()
        print('backward')
        time.sleep(1)

    GPIO.cleanup()
```

Accelerometer Driver

```python
import smbus
import math
import time

# registers for pwr mngmnt
pwr1 = 0x6b
pwr2 = 0x6c

#raw gyro registers
raw_gyro_x = 0x43
raw_gyro_y = 0x45
raw_gyro_z = 0x47

# raw accelerometer registers
raw_acc_x = 0x3b
raw_acc_y = 0x3d
raw_acc_Z = 0x3f

# default scaling values
default_acc_scaling_factor =  16384.0
default_gyro_scaling_factor = 131

# set smbus and i2c address
bus = smbus.SMBus(1) # or bus = smbus.SMBus(0) for Revision 1 boards
address = 0x68       # This is the address value read via the i2cdetect command

def read_byte(adr):
    return bus.read_byte_data(address, adr)

def read_word(adr):
    high = bus.read_byte_data(address, adr)
    low = bus.read_byte_data(address, adr+1)
    val = (high << 8) + low
    return val

def read_word_2c(adr):
    val = read_word(adr)
    if (val >= 0x8000):
        return -((65535 - val) + 1) #two's complement
    else:
        return val

def dist(a,b):
    return math.sqrt((a*a)+(b*b))
```

```
def dist(a,b):
    return math.sqrt((a*a)+(b*b))

def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)

def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)

# Wake up MPU 6050 (begins in sleep mode)
bus.write_byte_data(address, pwr1, 0)

# printing
while True:
    print "gyro data"
    print "---------"

    gyro_xout = read_word_2c(raw_gyro_x)
    gyro_yout = read_word_2c(raw_gyro_y)
    gyro_zout = read_word_2c(raw_gyro_z)

    print "gyro_xout: ", gyro_xout, " scaled: ", (gyro_xout / default_gyro_scaling_factor)
    print "gyro_yout: ", gyro_yout, " scaled: ", (gyro_yout / default_gyro_scaling_factor)
    print "gyro_zout: ", gyro_zout, " scaled: ", (gyro_zout / default_gyro_scaling_factor)
    print
    print "accelerometer data"
    print "------------------"

    accel_xout = read_word_2c(raw_acc_x)
    accel_yout = read_word_2c(raw_acc_y)
    accel_zout = read_word_2c(raw_acc_Z)

    accel_xout_scaled = accel_xout / default_acc_scaling_factor
    accel_yout_scaled = accel_yout / default_acc_scaling_factor
    accel_zout_scaled = accel_zout / default_acc_scaling_factor

    print "accel_xout: ", accel_xout, " scaled: ", accel_xout_scaled
    print "accel_yout: ", accel_yout, " scaled: ", accel_yout_scaled
    print "accel_zout: ", accel_zout, " scaled: ", accel_zout_scaled
    print "x rotation: " , get_x_rotation(accel_xout_scaled, accel_yout_scaled, accel_zout_scaled)
    print "y rotation: " , get_y_rotation(accel_xout_scaled, accel_yout_scaled, accel_zout_scaled)
    time.sleep(0.5)
```

MEASUREMENT DATA

Video Test:

- Parameters:
    - 30 frames per second
    - Resolution: 480 x 320
    - Capture Format: BGR
    - **1000 Trial Average Thread Runtime: 843.373 us**

```
# This code sets up a video feed from your system's default web cam
#cap = cv2.VideoCapture(0)
numSamples = 1000
sum = 0
avg = 0
sample = 0
# Capture rames from the camera
for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):

        start_time = time.time()
    # Read a frame from the camera
        image = frame.array

    # Display the frame in a window
    #cv2.imshow('frame', frame)
    #cv2.imshow('image', image)

        rawCapture.truncate(0)
        sample = sample + 1
        endTime = time.time() - start_time
        sum = sum + endTime

        if(sample == numSamples):
                avg = sum/numSamples
                print("\n\n--- Average is: %s seconds " % avg)
                break
    # This code will quit the program if 'q' is pressed
        if cv2.waitKey(1) & 0xFF == ord('q'):
                break

# Release system webcam and close all windows
cv2.destroyAllWindows()

time.sleep(1)

raise SystemExit
```

```
ideo_test.py

--- Average is: 0.000843373775482 seconds
pi@raspberrypi ~/Documents/computer_vision_car/
```

Accelerometer Test:

- Parameters:
  - I2C default speed of 100 kbps
  - MPU – 6050 Accelerometer + Gyro
  - Computation of scaled values for all 6 axes
  - **1000 Trial Average Thread Runtime: 6.051 ms**

```
# Wake up MPU 6050 (begins in sleep mode)
bus.write_byte_data(address, pwr1, 0)

sum = 0
avg = 0
numSamples = 1000
sample = 0
# printing
while True:

        start_time = time.time()

        gyro_xout = read_word_2c(raw_gyro_x)
        gyro_yout = read_word_2c(raw_gyro_y)
        gyro_zout = read_word_2c(raw_gyro_z)

        accel_xout = read_word_2c(raw_acc_x)
        accel_yout = read_word_2c(raw_acc_y)
        accel_zout = read_word_2c(raw_acc_Z)

        accel_xout_scaled = accel_xout / default_acc_scaling_factor
        accel_yout_scaled = accel_yout / default_acc_scaling_factor
        accel_zout_scaled = accel_zout / default_acc_scaling_factor

        endTime = time.time() - start_time
        sample = sample + 1
        sum = sum + endTime

        if(sample == numSamples):
                avg = sum/numSamples
                print("--- Average Time: %s seconds " % avg)
                break
```

```
pi@raspberrypi ~/Documents/computer_vision_car $ sudo python accelerometer_test.py
--- Average Time: 0.00605137085915 seconds
```

Observations:

It is still necessary to compute overhead for the PID controller thread, and for the computer vision data extraction subroutines to provide a rough estimate of the computational load distribution of the system.

Furthermore, it is worthy to note that the speed of the I2C bus can be increased four-fold to 400 kpbs for a possible added improvement of 2 ms in data acquisition overhead. It is important to consider adding a software circular buffer for acquiring data from the accelerometer + gyro, or simply to consider using a low polling frequency to further decrease the computational load on the BCM2835 processor.