# Monday, May 18, 15

Flow chart for path finder.

1. Find median = all < 400?
   1. Yes, FORWARD 10ms, GOTO 1
2. Median spread is balanced and >= 3 sensors wide?
   1. Yes, cross found, stop.
3. Median to right?
   1. Yes, RIGHT 10ms, GOTO 2
4. Median to left?
   1. Yes, Left 10ms, GOTO 1
5. All < 400?
   1. Yes, end, stop.

# Monday, May 11, 15

Flow chart for maze solver.

1a **New path**, Observation:

Narrow Centered line

Map: add centerline-traversed forward.

Go forward.

Goto 1a.

Line end

Map: mark deadend.

Dead end, rotate 180.

Goto 2a

Wide Line Both Sides, T or terminus

Set possible-T at current position.

Map: mark T-possible-terminus-traversed

Set depth = 1

Set line width = 1

Go forward.

Goto 3a

Wide Line Left

Rotate left 90°, backup sensor length.

Goto 4a

Wide Line Right

Rotate right 90°, backup sensor length.

Goto 5a

2a **Return path**, Observation:

3a **Disambiguating T-possible-terminus**, Observation:

Narrow Centered line, cross junction “+”

Map: mark (previous) possible-T position as cross junction., traversed from behind to forward.

Map has not traversed left =>

Backup ½ line width.

Rotate left 90°

Backup sensor length.

Goto 4a.

Map has not traversed right =>

Backup ½ line width.

Rotate right 90°

Backup sensor length.

Goto 5a.

Map has not traversed forward =>

Map: add centerline forward.

GO forward.

Line end

Wide Line Both Sides

Set depth += 1

If depth > exit depth, => 6a Terminus

Wide Line Left

Wide Line Right

4a **First decision after left turn**, Observation:

Narrow Centered line

Map: add centerline-traversed forward.

Goto 1a;

Line end

XXXXXXX

Wide Line Both Sides

Wide Line Left

Wide Line Right

5a **First decision after right turn**, Observation:

Narrow Centered line

Line end

Wide Line Both Sides

Wide Line Left

Wide Line Right

6a **Terminus,** Observation

# Tuesday, May 5, 15

Plan for designing a bot that can:

* Line follow.
* Solve a maze.
* Fine an object, move to touch it, and return.

Hardware features

* Motors.
* Odometry.
* Sonar. Possibly 4.
* LIDAR.
* 9 degree of freedom IMU.
* Camera.

Software features

* Integration of odometry, motor commands, gyros and magnetic compass to estimate position.
* LIDAR to build map, estimate position.
* SONAR to redundantly sample local obstacles.
* Accelerometer to measure tilt, prevent turnover.
* Path-map builder to track past movements.
* Goal-directed problem solver
* Camera feature recognition.
* Telepresence, pose estimation
* I2C communication between PI (master) and Arduino.
* USB between PI and IMU and LIDAR.
* Robot simulator.

# Monday, May 4, 15

Makefile

#DEBUG = -g -O0

DEBUG = -O3

CC = gcc

INCLUDE = -I/usr/local/include -I/Users/michaelwimble/Desktop/RobotV1/wiringPi/wiringPi

CFLAGS = $(DEBUG) -Wall $(INCLUDE) -Winline -pipe

LDFLAGS = -L/usr/local/lib

LDLIBS = -lwiringPi -lwiringPiDev -lpthread -lm

###############################################################################

SRC = \

MotorDriver.cpp \

MotorTest.cpp \

SetupGpio.cpp

OBJ = $(SRC:.cpp=.o)

BINS = $(SRC:.cpp=)

all: $(BINS)

robotv1: $(OBJ)

@echo [link]

@$(CC) -o $@ robotv1.o $(LDFLAGS) $(LDLIBS)

motorTest: ${OBJ}

@echo [motorTest]

@$(CC) -O $@ motorTest.o $(LDFLAGS) $(LDLIBS)

.cpp.o:

@echo [CC] $<

@$(CC) -c $(CFLAGS) $< -o $@

clean:

@echo "[Clean]"

@rm -f $(OBJ) \*~ core tags $(BINS)

tags: $(SRC)

@echo [ctags]

@ctags $(SRC)

depend:

makedepend -Y $(SRC)

# DO NOT DELETE

MotorDriver.h

#ifndef \_\_MOTORDRIVER\_H\_\_

#define \_\_MOTORDRIVER\_H\_\_

#include <stdint.h>

#include "wiringPi.h"

/\*\*\*\*\*\*Pins definitions\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define MOTORSHIELD\_IN1 0

#define SPEEDPIN\_A 2

#define SPEEDPIN\_B 3

#define MOTORSHIELD\_IN2 12

#define MOTORSHIELD\_IN3 13

#define MOTORSHIELD\_IN4 14

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*Motor ID\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define MOTORA 0

#define MOTORB 1

#define MOTOR\_POSITION\_LEFT 0

#define MOTOR\_POSITION\_RIGHT 1

#define MOTOR\_CLOCKWISE 0

#define MOTOR\_ANTICLOCKWISE 1

#define USE\_DC\_MOTOR 0

struct MotorStruct

{

int8\_t speed;

uint8\_t direction;

uint8\_t position;

};

/\*\*Class for Motor Shield\*\*/

class MotorDriver

{

MotorStruct motorA;

MotorStruct motorB;

public:

MotorDriver();

void init();

void configure(uint8\_t position, uint8\_t motorID);

void setSpeed(int8\_t speed, uint8\_t motorID);

void setDirection(uint8\_t direction, uint8\_t motorID);

void rotate(uint8\_t direction, uint8\_t motor\_position);

void rotateWithID(uint8\_t direction, uint8\_t motorID);

void goForward();

void goBackward();

void goLeft();

void goRight();

void stop();

void stop(uint8\_t motorID);

};

extern MotorDriver motordriver;

#endif

MotorDriver.cpp

// Author:Frankie.Chu

// Date:20 November, 2012

//

// This library is free software; you can redistribute it and/or

// modify it under the terms of the GNU Lesser General Public

// License as published by the Free Software Foundation; either

// version 2.1 of the License, or (at your option) any later version.

//

// This library is distributed in the hope that it will be useful,

// but WITHOUT ANY WARRANTY; without even the implied warranty of

// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU

// Lesser General Public License for more details.

//

// You should have received a copy of the GNU Lesser General Public

// License along with this library; if not, write to the Free Software

// Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

//

// Modified record:

//

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "MotorDriver.h"

#include "SetupGPIO.h"

MotorDriver::MotorDriver() {

SetupGPIO dummy;

}

void MotorDriver::init() {

stop();

/\*Configure the motor A to control the wheel at the left side.\*/

configure(MOTOR\_POSITION\_LEFT,MOTORA);

/\*Configure the motor B to control the wheel at the right side.\*/

configure(MOTOR\_POSITION\_RIGHT,MOTORB);

setSpeed(127,MOTORA);

setSpeed(127,MOTORB);

setDirection(MOTOR\_ANTICLOCKWISE,MOTORA);

setDirection(MOTOR\_CLOCKWISE,MOTORB);

}

void MotorDriver::configure(uint8\_t position, uint8\_t motorID) {

if(motorID == MOTORA)motorA.position = position;

else motorB.position = position;

}

void MotorDriver::setSpeed(int8\_t speed, uint8\_t motorID) {

if(motorID == MOTORA) motorA.speed = speed;

else if(motorID == MOTORB) motorB.speed = speed;

}

void MotorDriver::setDirection(uint8\_t direction, uint8\_t motorID) {

if(motorID == MOTORA)motorA.direction= direction;

else if(motorID == MOTORB)motorB.direction = direction;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*Function: Get the motor rotate \*/

/\*Parameter:-uint8\_t direction,Clockwise or anticlockwise; \*/

/\* -uint8\_t motor\_position,MOTOR\_POSITION\_LEFT or \*/

/\* MOTOR\_POSITION\_RIGHT; \*/

/\*Return: void \*/

void MotorDriver::rotate(uint8\_t direction, uint8\_t motor\_position) {

if(motor\_position == motorA.position) {

rotateWithID(direction,MOTORA);

}

if(motor\_position == motorB.position) {

rotateWithID(direction,MOTORB);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*Function: Get the motor rotate \*/

/\*Parameter:-uint8\_t direction,Clockwise or anticlockwise; \*/

/\* -uint8\_t motor\_position,MOTORA or MOTORB \*/

/\*Return: void \*/

void MotorDriver::rotateWithID(uint8\_t direction, uint8\_t motorID) {

uint8\_t in1\_level,in2\_level;

if(MOTOR\_CLOCKWISE == direction) {

in1\_level = LOW;

in2\_level = HIGH;

} else {

in1\_level = HIGH;

in2\_level = LOW;

}

if(motorID == MOTORA) {

analogWrite(SPEEDPIN\_A,motorA.speed);

digitalWrite(MOTORSHIELD\_IN1,in1\_level);

digitalWrite(MOTORSHIELD\_IN2,in2\_level);

} else if(motorID == MOTORB) {

analogWrite(SPEEDPIN\_B,motorB.speed);

digitalWrite(MOTORSHIELD\_IN3,in1\_level);

digitalWrite(MOTORSHIELD\_IN4,in2\_level);

}

}

void MotorDriver::goForward() {

rotate(MOTOR\_ANTICLOCKWISE,MOTOR\_POSITION\_LEFT);

rotate(MOTOR\_CLOCKWISE,MOTOR\_POSITION\_RIGHT);

}

void MotorDriver::goBackward() {

rotate(MOTOR\_ANTICLOCKWISE,MOTOR\_POSITION\_RIGHT);

rotate(MOTOR\_CLOCKWISE,MOTOR\_POSITION\_LEFT);

}

void MotorDriver::goLeft() {

rotate(MOTOR\_CLOCKWISE,MOTOR\_POSITION\_RIGHT);

rotate(MOTOR\_CLOCKWISE,MOTOR\_POSITION\_LEFT);

}

void MotorDriver::goRight() {

rotate(MOTOR\_ANTICLOCKWISE,MOTOR\_POSITION\_RIGHT);

rotate(MOTOR\_ANTICLOCKWISE,MOTOR\_POSITION\_LEFT);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void MotorDriver::stop() {

/\*Unenble the pin, to stop the motor. \*/

digitalWrite(SPEEDPIN\_A,LOW);

digitalWrite(SPEEDPIN\_B,LOW);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void MotorDriver::stop(uint8\_t motorID) {

if(motorID == MOTORA)digitalWrite(SPEEDPIN\_A,LOW);

else if(motorID == MOTORB)digitalWrite(SPEEDPIN\_B,LOW);

}

MotorDriver motordriver;

SetupGPIO.h

#ifndef \_\_SetupGPIO\_H\_\_

#define \_\_SetupGPIO\_H\_\_

class SetupGPIO {

public:

SetupGPIO();

private:

static bool initialized;

};

#endif

SetupGPIO.cpp

#include "MotorDriver.h"

#include "SetupGPIO.h"

#include <wiringPi.h>

SetupGPIO::SetupGPIO() {

if (!initialized) {

wiringPiSetupGpio();

pinMode(MOTORSHIELD\_IN1, OUTPUT) ;

pinMode(SPEEDPIN\_A, OUTPUT) ;

pinMode(SPEEDPIN\_B, OUTPUT) ;

pinMode(MOTORSHIELD\_IN2, OUTPUT) ;

pinMode(MOTORSHIELD\_IN3, OUTPUT) ;

pinMode(MOTORSHIELD\_IN4, OUTPUT) ;

initialized = true;

}

}

bool SetupGPIO::initialized = false;

MotorTest.cpp

#include "MotorDriver.h"

void setup()

{

/\*Configure the motor A to control the wheel at the left side.\*/

/\*Configure the motor B to control the wheel at the right side.\*/

motordriver.init();

motordriver.setSpeed(200,MOTORB);

motordriver.setSpeed(200,MOTORA);

}

void loop()

{

motordriver.goForward();

delay(2000);

motordriver.stop();

delay(1000);

motordriver.goBackward();

delay(2000);

motordriver.stop();

delay(1000);

motordriver.goLeft();

delay(2000);

motordriver.stop();

delay(1000);

motordriver.goRight();

delay(2000);

motordriver.stop();

delay(1000);

}

int main() {

loop();

return 0;

}