Lab 3 — Turning Algorithm

This lab is designed to guide you through developing the turning algorithm using the IR sensors to detect when a turn is finished executing and to provide information for how to start your research option for lab 3.

You can find this and future lab updates at http://www.csulb.edu/~hill

Table of Contents

What is New? 1
Data Types 1
C++ Code 1
Arudino Built-In Functions 2
Developing the Turning Algorithm 2
Methods for Turning 2
Timed Turning 2
Finite State Machine Turning 3
Lab 3 Research Option 4
Lab 3 Deliverable(s) 4
Checklist 4

What is New?

New terminology and concepts are listed below in black. If you have any questions on this information, refer back to the lectures on the introduction to C++ and configuring the GPIO registers.

Data Types

```
const        // Define variable as a constant
static       // Define variable that will persist after a function return
uint8_t      // Defines the data type as an unsigned 8 bit integer (0-255)
uint16_t     // Defines the data type as an unsigned 16 bit integer (0-65,535)
```

C++ Code

```
#define Name Number // Allows the user to define Name to be equivalent to Number
#include <filename> // Used to include any built-in libraries such as avr/io.h
#include “filename” // User defined libraries or files that are in the sketch folder
```

Bit / Byte Operations

```
variable = byteValue << numberOfShifts; // Left shift byte value by the number defined
variable = _BV (bitNumber);             // Turns a bit number into a byte value
variable |= _BV(bitNumber);             // Set a specific bit in a byte value
variable &= ~_BV(bitNumber);            // Clear a specific bit in a byte value
variable = byteValue | byte Value;      //
```

Arudino Built-In Functions

Pin Configuration & Control

```
pinMode(pin_number, TYPE);        // Define a pin as an input or output
digitalRead(pin_number);          // Read a digital value for an input pin
analogRead(pin_number);           // Read an analog value from an input pin
digitalWrite(pin_number, output); // Output a digital value to an output pin
analogWrite(pin_number, PWM_Value);
```

Developing the Turning Algorithm

From the previous labs, we are able to fully control the motors of your robot and have gotten an efficient line following algorithm implemented. In this lab, we are focusing on crossing intersections and turning the robot. When this lab is completed, you will be able to control your robot for all situations it can encounter in the maze and will be ready to develop the control algorithm for navigating your path through the maze in Labs 4, 5 and 6.

Methods for Turning

While there are many methods for executing a turn at an intersection, we will be focusing on two. The first is using time to determine when to stop and the second is developing a finite state machine that will accurately identify the position of the robot. We will go through both and hopefully be able to implement the finite state machine solution.

Timed Turning

Implementing the timed turning method is very straightforward. It will require a lot of trial and error to figure out the exact amount of time needed to get the robot to turn left, right, or around. It also requires being familiar with the specific characteristics of your robot since the motor speeds of a different robot may not work for the time that you found. However, once you have done all of this work, your robot should be able to execute exact turns every single time.

There are two problems with this solution. First, it does not involve the IR sensors in any way to detect the position of the robot, and is therefore an “open-loop” solution. Put another way, the robot has no way of knowing if anything is actually happening. Second, the solution is unique to your implementation. In the next section, you will develop a “closed-loop” algorithm that can be applied universally. The algorithm will be implemented by a finite state machine.

Finite State Machine Turning

The core concept behind the finite state machine solution is to translate the physical position of the robot when it is over the intersection to specific states that it will go through as it executes the turn. Some things to consider for this is what the initial position should be when the turn is executed. We can start at the moment the robot detects the intersection or after it has moved past the intersection. The initial position will determine the way the robot needs to turn in order to be aligned with the line after the turn.

Our first attempt at this solution will assume that the turn starts when the robot first detects the intersection. This means that all four IR sensors should be detecting the black line. If we are trying to handle a left turn, it should run the left motor at the minimum speed and the right motor at full speed. That way the robot will arc towards the left and should align with the line after the turn. There will be several transitions that the IR sensors should detect as the robot takes this path and we can associate each with a different state for the finite state machine.

The following algorithm assumes a line following and not an edge following robot with four IR sensors. If you are using the 3DoT IR Shield (Figure 1) the outer sensors use analog inputs and may be used with the AnalogRead() function to more accurately detect when a transition has occurred.

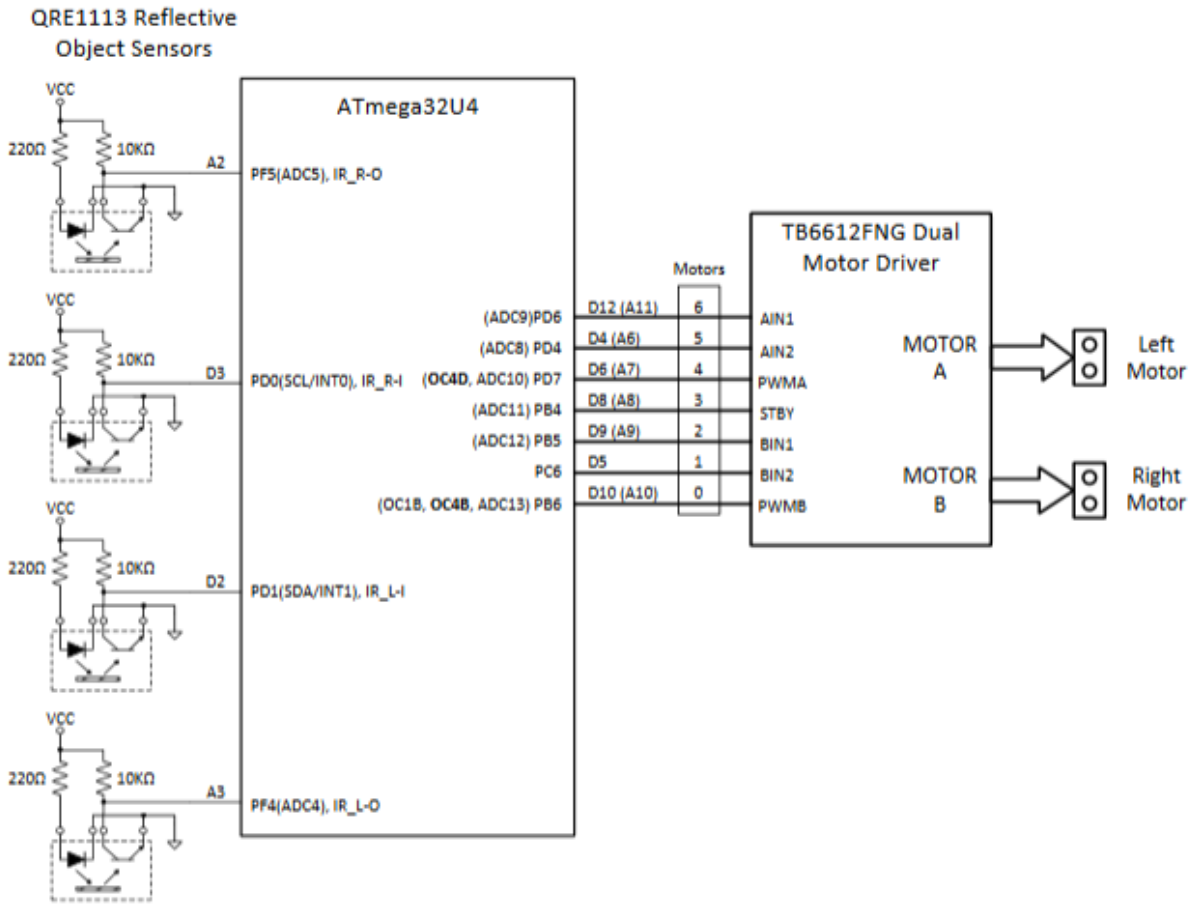

Figure 1 – Optical Sensor to Motor Driver Interface

The first state will be when the two outer sensors detect the black line of the intersection. The second state will be when the outer sensor on the right side leaves the black line and hits the white space. To make things easier to detect. During this time, the left outer sensors should also be transitioning to the white space as well. Eventually, the outer sensors will hit the vertical line and that would indicate that the turn is complete.

Apply what we discussed in lab to develop this finite state machine and verify that it works as intended. Please note any changes or modifications that you needed to make.

Lab 3 Research Option

Now that we have covered how the turning can be done with the IR sensors, you should focus on implementing the research option that you signed up for at the beginning of the semester. This could be incorporating a rotary encoder and a different sensor to improve the turning algorithm. Because of the wide range of options for this, we do not have much to discuss besides making sure that you provide a write up of how you developed the new algorithm.

Lab 3 Deliverable(s)

All labs should represent your own work - DO NOT COPY.

Submit all of the files in your sketch folder. Make sure that the code compiles without any errors.

Checklist

Remember before you turn in your lab...

1. Timed sketch with comments. Highlight times experimentally determined for your robot.
2. FSM sketch with comments.
3. The sketch of your own implementation.