Approach Indicator Direction Sensing

Contents

The Project

Many railroad interlocking towers had alarm bells to alert the tower operator that a train was approaching. On a large club railroad where I volunteer there are train order offices located under the

layout. We set up a closed-circuit camera system so that the agent/operator can see the approaches to their town. We also wanted to implement alerts when a train approached.

We decided to implement an Arduino-based solution to the problem. Arduinos are a low-cost, highly programmable solution for analog and digital sensor-based projects.

Two Sketches: ApproachIndicator and HelpInstallAI

There are two sketches involved. ApproachIndicator is the primary sketch which provides the approach notification function. HelpInstallAI is a supporting sketch which will assist in determining/confirming that sensors are connected properly.

Sketch: ApproachIndicator

Features

The software provides:

- Up to four sets (eight approaches) of location sensors per Arduino Nano.
- Each approach consists of 2 analog or digital sensors.
- Each approach drives an individual or shared alert.
- Two approaches are usually combined into a "station."
- Analog sensors are automatically and periodically recalibrated.

The Design

The design is based on having two sensors at each end of a town. These *home* and *distant* sensor sets for the *east* and *west* (or north/south or left/right) ends of town will be scanned and timed such that we can determine the direction of travel of the train. If the distant sensor is triggered, and subsequently (or at the same time) the home sensor is triggered we know that the train is approaching and the alert should be sounded.

The Arduino has the ability to sense both analog and digital signals. Accordingly our design allows for either. Our design can drive different alerts based on whether the east or west sensors have been triggered. We also use the Arduino sketch to auto-calibrate the analog sensors.

The software for the Arduino is very configurable by setting control constants in the head of the program.

Placing Sensors

The sensors are placed in pairs, each pair having a *home* and a *distant* sensor. The spacing of the sensors must be such that both sensors must be covered for the alert to be generated. This means that if a single locomotive is to trigger the alert, the sensors must be placed closer to each other than the length of the locomotive. Each sensor pair must be of the same type, either analog or digital. An Arduino can read up to 8 analog sensors depending on the type of Arduino. (Uno supports 6, Nano supports 8.) The digital sensors can be configured to read up to 8 digital sensor inputs.

Track diagram for a town or tower.

| East | distant | home | Tower | home | distant | West |
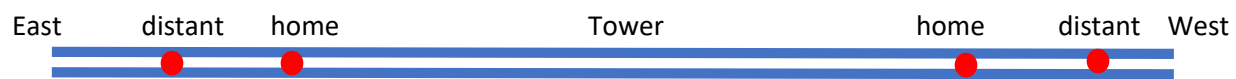
The distance between the distant and home sensors is also dependent on the timing which is configured in the Arduino program. If the program is not sampling the sensors frequently enough it could miss the passage of a fast-moving small engine or short train.

Sensor Sets [Software version 5.1]

Sensors are defined as approaches. A "station" would normally be a town or tower and would have connections to two approaches (four sensors) of the same type, analog or digital. The program is set up for four stations per Arduino. Each approach can be assigned its own digital output for driving an alert.

```
// Define the active stations: 0-1 Analog detection, 2-3 digital detection
// A "station" is a set of two approaches, four sensors:
//                 East DISTANT/HOME and West DISTANT/HOME
//                 Station 0   Station 1      Station 2   Station 3
//                 East  West  East  West   East  West   East   West
const boolean WorkApproach [8] =
                 {true, true, false,false, false,false,  false, false
```

Sensors-Analog

Light detecting devices, either photo-transistors or photo-resistors may be used. They are connected to the Arduino's analog ports. The design calls for pull-down resistors on the ground side for each sensor.

Sensors-Digital

A digital signal (high/low) can come from an infrared detector or some such.

Software: Sensors

Here are the definitions for the analog sensors:

```
// Analog sensor pins
// NOTE a sensor entry of 0 in the *pins arrays will be skipped
//                 Station 0       Station 1
//                 East   West    East   West
//                 D H    D H     D H    D H
const int Apins[8] = {A0,A1,  A2,A3,   A4,A5,  A6,A7};// analog sensors - Nano
```

Note the order of the home versus distant sensors. This ordering must be followed or the Apins array must be adjusted.

An unexpected feature showed up during on-layout testing. Everything worked fine with the breadboard in my office. However, the readings were all over the map when I tested the analog sensors on the layout. An experienced colleague suggested that the sensors were being affected by the LED lighting in the layout room, and recommended integrating readings over a power cycle, i.e. 34 milliseconds for the 60 hertz mains. The numbers below control the integration loop and are in milliseconds.

```
// analog sampling controls - needed to average light flicker
// length of sample integration in millisecs
// 34 ms for 60 cycle lighting mains
// 40 ms for 50 cycle lighting mains
#define LoopMS 34
// delay within integration loop in millisecs 0 = no delay
#define DlyInLoop 0
```

The software automatically calibrates the sensors and keeps track of high/low values for each. I built in a periodic recalibration for the analog sensors in case sunlight or other conditions changed the light hitting the sensors.

```
// minutes between recalibrate low/high analog sensor values
const int RecalibrateMts = 5;
```

The digital sensors are more straightforward and are defined below. The InvertDig flag is used if the sensor returns a value the opposite of required.

```
// Digital sensor pins
//                   Station 2   Station 3
//                   East  West  East  West
//                   D  H  D  H  D  H  D  H
const int Dpins[8] = {2, 3, 4, 5, 6, 7, 8, 9};// digital sensors
//
// invert flag for digital sensor if HIGH sensor means clear
boolean   InvertDig = true;
```

Sensor setup: When installing or debugging an installation, I found a need to confirm that the Arduino was detecting the sensor properly. I decided to use the on-board LED 13 for this purpose. You can choose some other digital output for this purpose. A 0 (zero) will disable this feature. In addition, refer to the HelpInstallAI sketch (below) for additional help during the install process.

```
// if SnsrActiveLED is non-zero, light the LED indicated if any sensor is active
#define SnsrActiveLED 13
// #define SnsrActiveLED 0
```

The main processing loop can also be timed. A non-zero time is normally only for debugging.

```
// delay in ms for main loop sampling
// -- 0 = continuous
#define MainLoopDlyMS 0
```

When I tested on the layout, I used a train with logging cars and a very short logging caboose. This caused the sensors to alternate between clear and occupied while the train was passing. I decided to place a timer to hold a sensor *occupied* (in software) to control wild swings caused by skeleton cars or couplings.

```
// hold occupied status for this time to minimize bounce (false clear)
// - for couplers, skeleton cars - in milliseconds
#define HoldOccMs 1000
```

Alerts

The software is set to generate alerts for each set. Those pins can drive a relay for driving a horn or bell. If you want to use a single relay, configure the constants below to use the same pin.

```
// digital output pins for alert relays for each approach
//                   Station 0   Station 1    Station 2   Station 3
//                   East  West  East  West   East  West  East  West
const int AlrtPins[8] =
                     {10,   10,   11,   11,    12,   12,   13,   13};
```

Make sure that these pins don't overlap with the digital sensor pins.

We also have definitions for the length of an alert, and for a timeout before a subsequent alert can occur on that pin.

```
// duration of alert in seconds - how long to ring the bell
const int AlertLenSecs[8] =
                    {2,2,2,2,2,2,2,2};

// dont repeat alert for a bit - in seconds - stop false repeats
#define DontRepeatSecs 30
```

Timer Discussion

Timers are used in the code throughout.

`MainLoopDlyMS` is the most important, since it tells you how often the sensors will be sampled. Make it too slow you will miss a fast-moving engine. This also relates to the distance between your home and distant sensors and how fast the trains run. Normally 0;

`LoopMS` and `DlyInLoop` control the integration loops for each set of analog sensors.

`HoldOccMs` and `DontRepeatSecs` block processing of new alerts. `DontRepeatSecs` if too large could block the alert of an engine or train following too closely to a prior train. It operates on a set basis. Note that it will not block an alert on the same set coming from a different direction. `HoldOccMs` does a similar function on a sensor-by-sensor basis. They are both intended to minimize falsely-repeating alerts.

Arduino Devices

Analog pins A0 thru A7: Phototransistors or Photoresistors may be used. The software depends on a pulldown to ground.

Digital pins 2 thru 9: Input: Any digital device providing an Arduino-compatible high/low may be used. There is a software flag `InvertDig` to reverse the meaning of the signal if necessary.

Digital pins 10 thru 13: Output: These pins are used to drive an alert per set.

Board Schmatic

There is a separate document ApproachIndicatorSchematic.docx which shows the layout of my wire-wrap prototype board.

MRCS Board

Model Railroad Control Systems offers a PCB for this project Contact them at https://www.modelrailroadcontrolsystems.com/   .

Sketch: HelpInstallAI

Features

There is an additional sketch which may be used to assist in installation of Approach Indicator. The sketch will drive the board to produce an output signal when a sensor is triggered. This way one can confirm that a sensor is a) working, and b) connected to the proper input.

The HelpInstallAI sketch will:
- First, trigger each defined output. We recommend that the outputs be connected to an audio device.
- Loop continuously, checking each defined set and its associated sensors.
- If a sensor is triggered, the sketch will trigger the associated output a number of times:
  - Report the set number +1 with a longer alert sequence
  - Report the line number +1 with a shorter alert sequence

Configuration

Here are the basic definitions within the sketch that may need modification:

```
// NOTE: The program may be configured to monitor 4 sensor sets/towers/bells
//       This would require 2 sets of analog and 2 sets of digital sensors
boolean UseSet [4] = {true,true,false,false};
//
// digital output pins for alerts for each set
const int Alrts[4] = {10,11,12,13};
//
// duration of alerts in milliseconds
#define SetAlertMs  1000
#define LineAlertMs  400
//
// Analog sensor pins
// NOTE a sensor entry of 0 in the *pins arrays will be skipped
//             Set0  ND NN SD SN  Set1 ND NN SD SN
const int Apins[8] = {A0,A1,A2,A3,    A4,A5,A6,A7};// analog sensors - Nano
//
// Digital sensor pins
//             Set2  ND NN SD SN  Set3 ND NN SD SN
const int Dpins[8] = { 2, 3, 4, 5,     6, 7, 8, 9};// digital sensors
// invert flag for digital sensor if HIGH sensor means clear
boolean  InvertDig = true;
//
// analog sampling controls - needed to average light flicker
// length of sample integration in millisecs
// 34 ms for 60 cycle lighting mains
// 40 ms for 50 cycle lighting mains
#define LoopMS 34
//
```

```
// analog sensors are recalibrated frequently to account for changes in lighting
//  such as sunlight moving
// minutes between recalibrate low/high analog sensor values
#define RecalibrateMts 5
//
// delay in ms for main loop sampling
// -- 0 = continuous
#define MainLoopDlyMS 5000
//
// echo to serial monitor
#define Trace 1
```