Approach Indicator Direction Sensing

Contents

Approach Indicator Direction Sensing 1
The Project 1
Two Sketches: ApproachIndicator and HelpInstallAI 2
Sketch: ApproachIndicator 2
Features 2
The software provides: 2
The Design 2
Placing Sensors 2
Sensors Sets [Software version 4.5] 3
Sensors-Analog 3
Sensors-Digital 3
Software: Sensors 3
Alerts 4
Timer Discussion 5
Arduino Devices 5
Board Schmatic 5
MRCS Board 5
Flowchart 6
Setup 6
GetDiff 9
GetAlog 10
GetDigI 11
Sketch: HelpInstallAI 12
Features 12
Configuration 12

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 are setting up a closed-circuit camera system so that the agent/operator can see the approaches to his/her 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 of location sensors per Arduino.
- Each set consists of 2 to 4 analog or digital sensors.
- Each set drives an individual or shared alert.
- Analog sensors are automatically and periodically calibrated.

The Design

The design is based on having two sensors at each end of a town. These *near* and *distant* sensor sets for the *north* and *south* (or east/west 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 near 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 north or south sensors have been triggered. We use the Arduino software to auto-calibrate the analog sensors.

We finalized on a design which will allow either analog or digital sensors. With the correct sensor configuration up to 4 towns can be configured on a single Arduino, each with its own alert.

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 *near* 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.

The distance between the distant and near 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.

Sensors Sets [Software version 4.6]

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

```
// Define the active sets: 0-1 Analog sensors, 2-3 digital sensors
// NOTE: The Uno only has 6 analog pins, so set 2 MUST be set false
//       The Nano has 8 pins so analog set 2 is an option
//                       Analog       Digital
//                    Set0   Set1   Set2   Set3
boolean UseSet [4] = {true,  true, false, false};
//
// 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,true,true};
// NOTE: IF only one set is to be used, update UseSet accordingly
//            or else false positives may result
// boolean UseSet [4] = {true,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. We use the pull-down to detect if the analog pin is connected to a 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:

```
// NOTE relative positioning of sensors
//    - pins and connections must match the order below
//NoD = 0 north distant sensor
//NoN = 1 north near sensor
//SoD = 2 south distant sensor
//SoN = 3 south near sensor

// 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
```

Note the order of the near 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
const int LoopMS = 34; // length of sample integration in millisecs
const int DlyInLoop = 0;// delay within integration in millisecs if desired
```

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
//             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;
```

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 and hold defines
// 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 5000
```

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 set
const int Alrts[4] = {10,11,10,11};
// or use this for example if only one output for all sets
```

```
// const int Alrts[4] = {10,10,10,10};
// or use this for example if 4 sets active with 4 bells
// const int Alrts[4] = {10,11,12,13}; // Note use of pin 13 - onboard LED
```

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 milliseconds - how long to ring the bell
#define AlertLenMs 4000
// dont repeat alert for a bit - in seconds - stop false repeats
#define DontRepeatSecs 10
```

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, such as 1 per second, and you will miss a fast-moving engine. This also relates to the distance between your near and distant sensors and how fast the trains run. Normally 0;

`LoopMS` and `DlyInLoop` control the integration loops for each set of analog sensors. This time is included in the timing of `CyclesPerSec`. Making these too large could effectively override `CyclesPerSec` since the data collection could take longer than the requested loop time.

`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 in order to detect the presence of the device if doing auto-detect.

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.

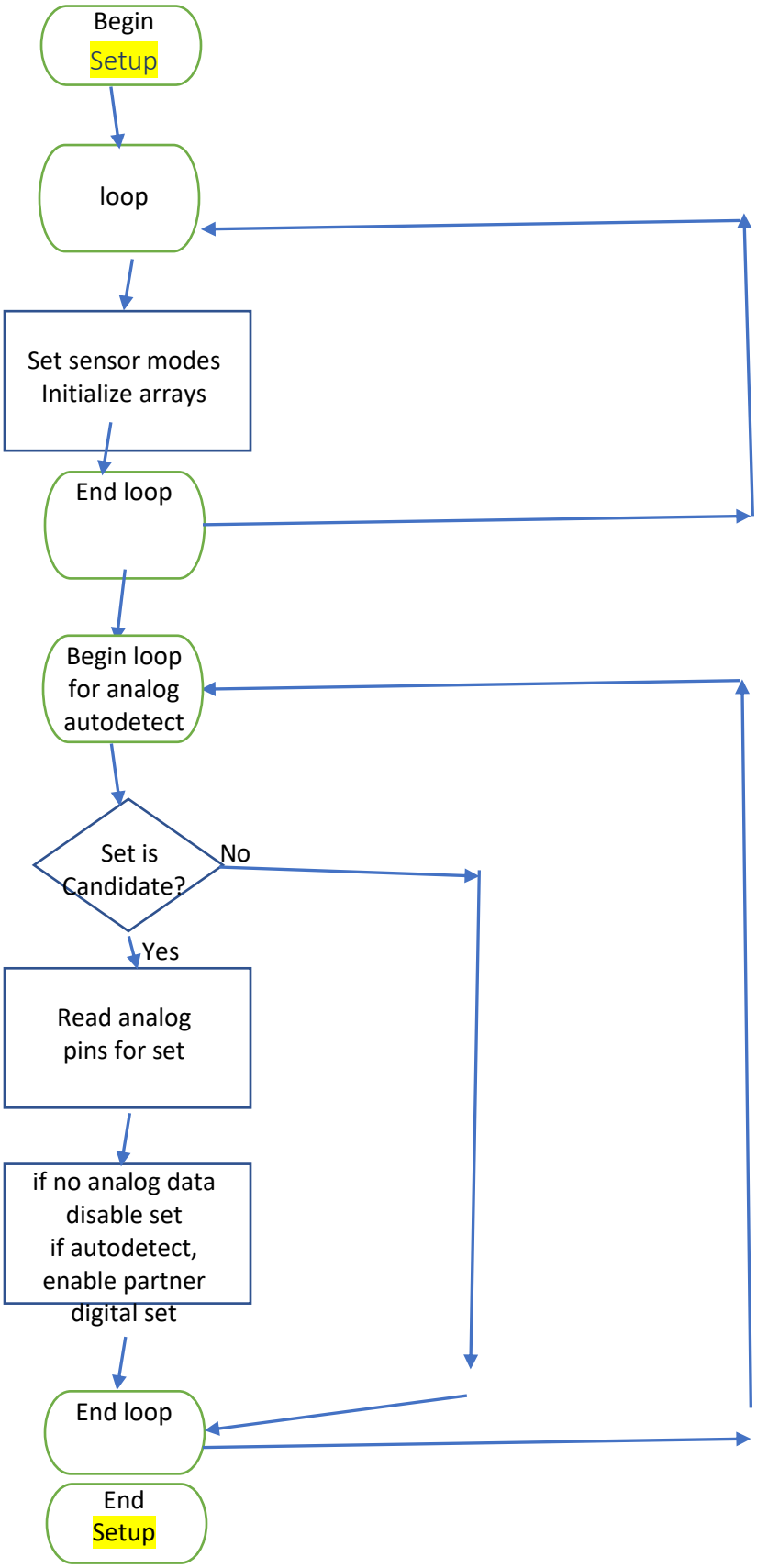
Board Schmatic

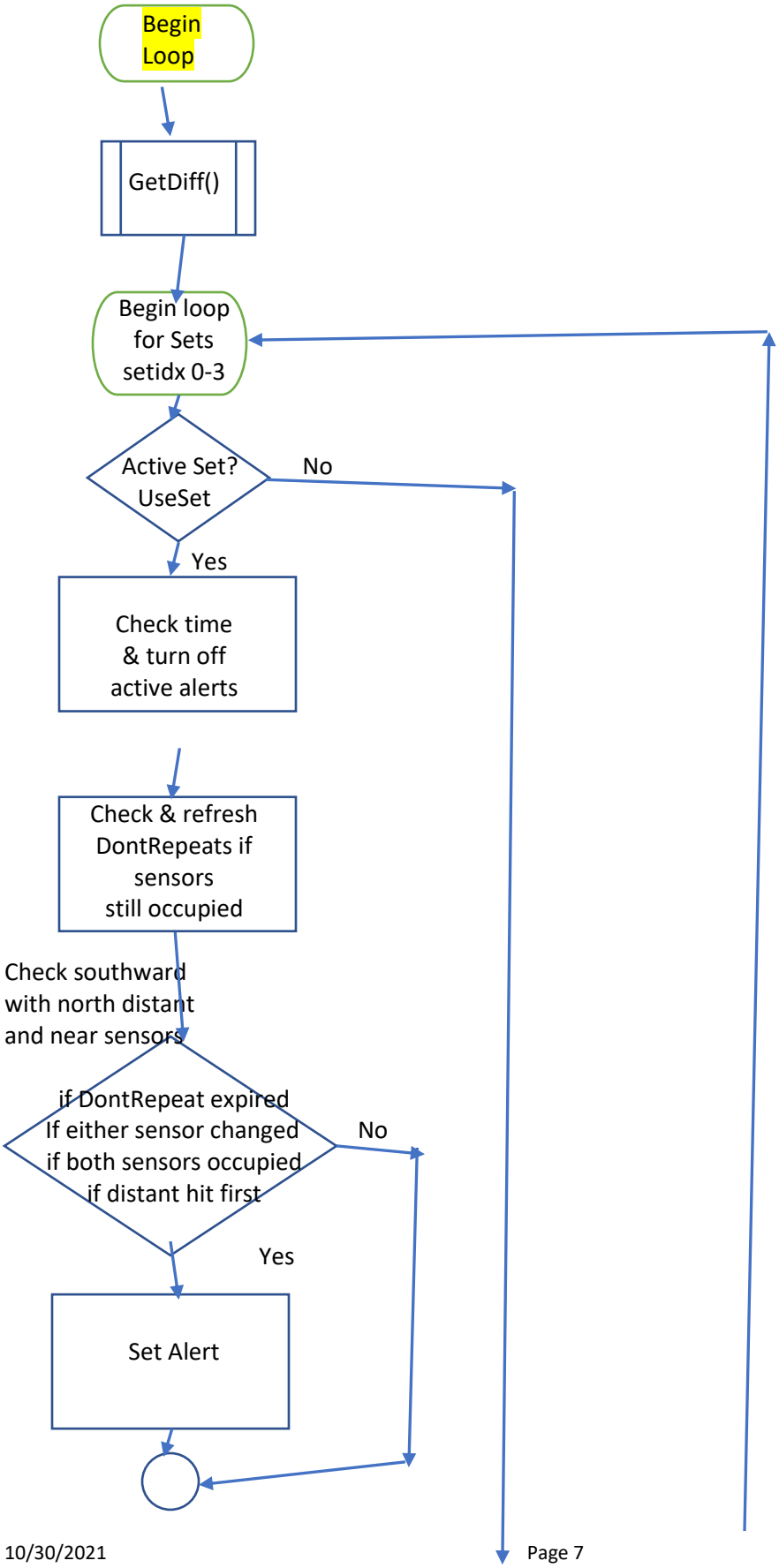
There is a separate document ApproachIndicatorSchematic.docx which shows the layout of my wire-wrap prototype board.

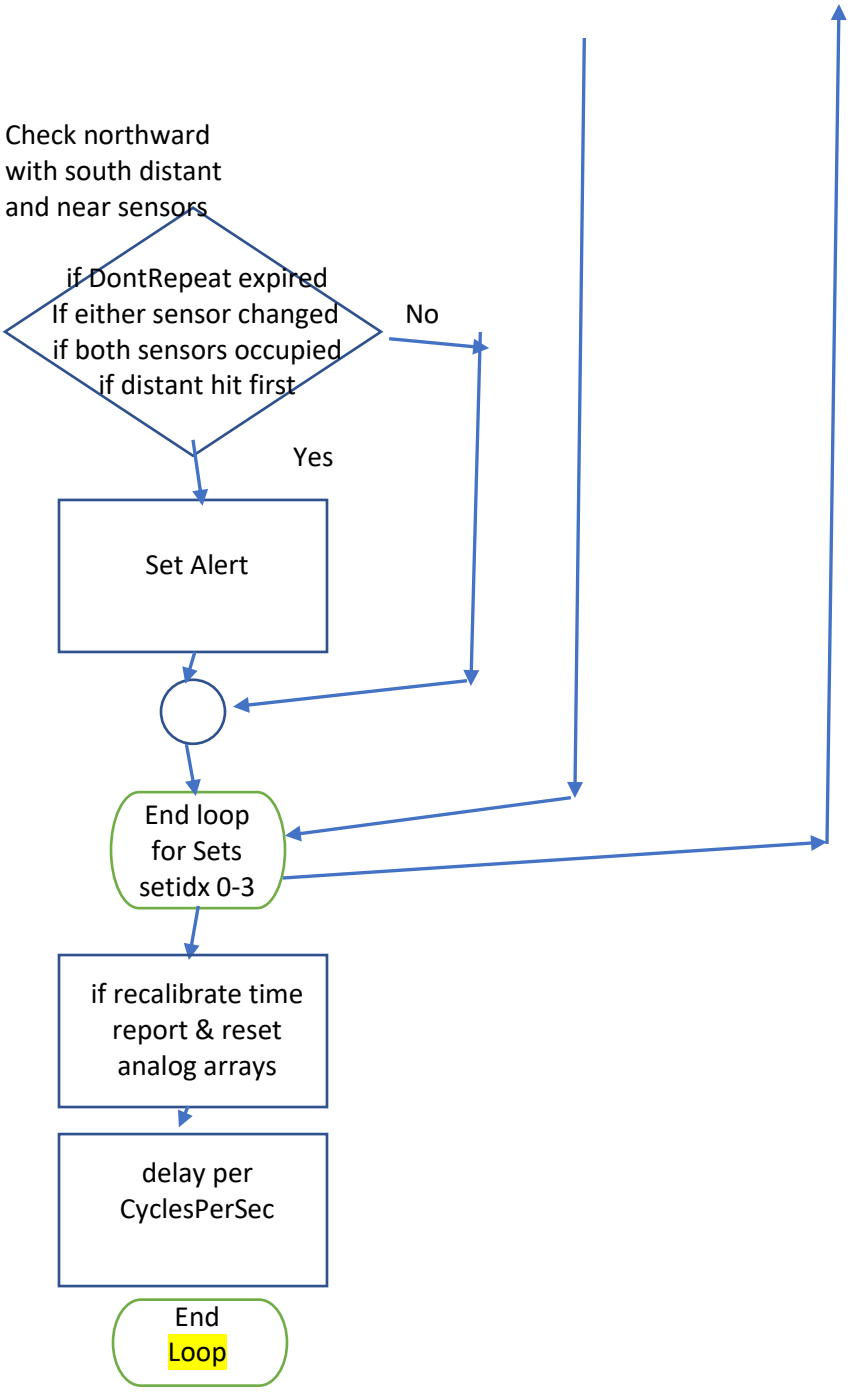
MRCS Board

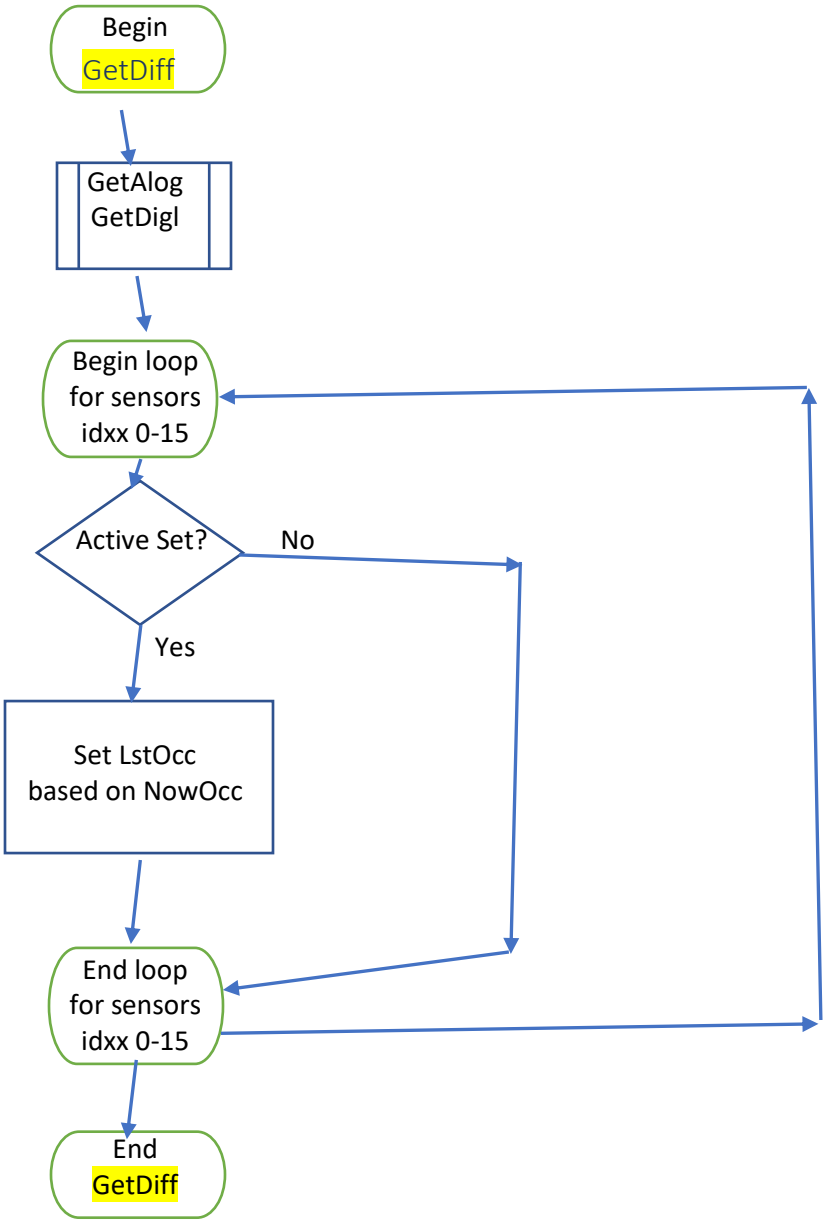
Model Railroad Control Systems offers a PCB for this project Contact them at https://www.modelrailroadcontrolsystems.com/ .

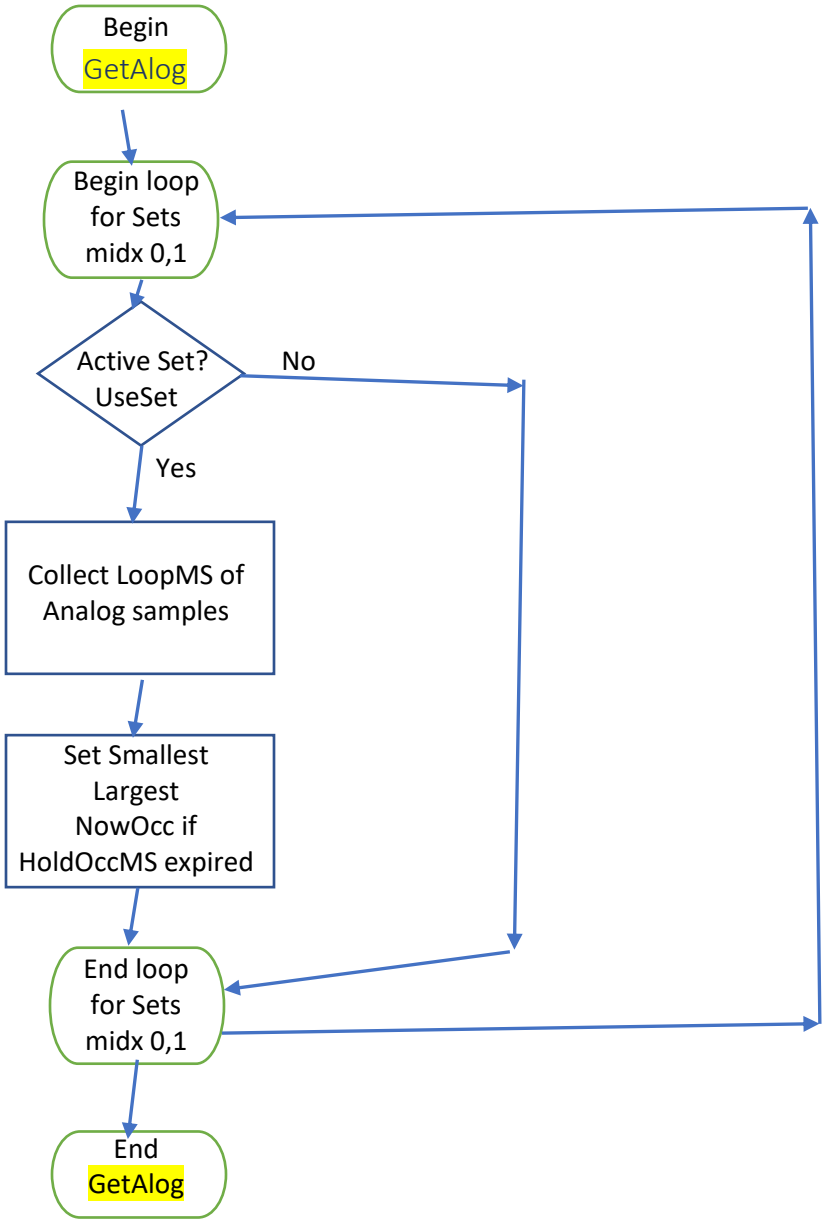
Flowchart

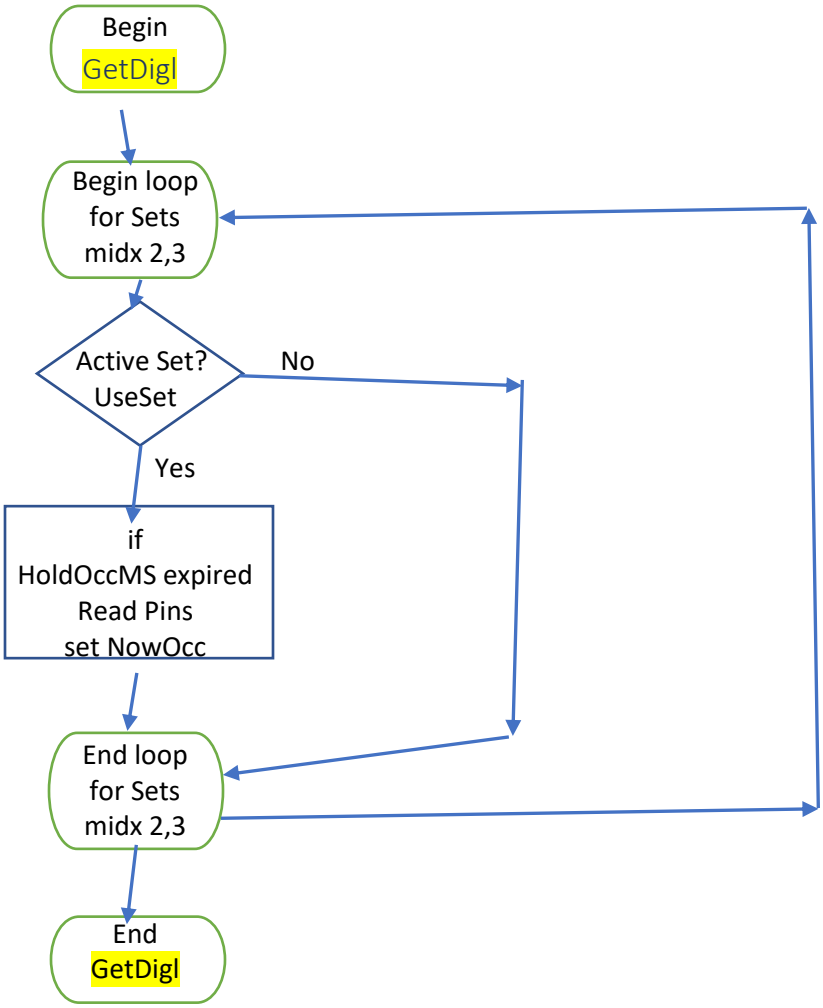

Begin
Setup
loop
Set sensor modes
Initialize arrays
End loop
Begin loop
for analog
autodetect
Set is
Candidate?
No
Yes
Read analog
pins for set
if no analog data
disable set
if autodetect,
enable partner
digital set
End loop
End
Setup


Begin Loop
GetDiff()
Begin loop for Sets setidx 0-3
Active Set? UseSet
No
Yes
Check time & turn off active alerts
Check & refresh DontRepeats if sensors still occupied
Check southward with north distant and near sensors
if DontRepeat expired
If either sensor changed
if both sensors occupied
if distant hit first
No
Yes
Set Alert


Check northward
with south distant
and near sensors
if DontRepeat expired
If either sensor changed
if both sensors occupied
if distant hit first
No
Yes
Set Alert
End loop
for Sets
setidx 0-3
if recalibrate time
report & reset
analog arrays
delay per
CyclesPerSec
End
Loop


Begin
GetDiff
GetAlog
GetDigl
Begin loop
for sensors
idxx 0-15
Active Set?
No
Yes
Set LstOcc
based on NowOcc
End loop
for sensors
idxx 0-15
End
GetDiff


Begin
GetAlog
Begin loop
for Sets
midx 0,1
Active Set?
UseSet
No
Yes
Collect LoopMS of
Analog samples
Set Smallest
Largest
NowOcc if
HoldOccMS expired
End loop
for Sets
midx 0,1
End
GetAlog


Begin
GetDigl
Begin loop
for Sets
midx 2,3
Active Set?
UseSet
No
Yes
if
HoldOccMS expired
Read Pins
set NowOcc
End loop
for Sets
midx 2,3
End
GetDigl

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] = {12,12,13,13};
//
// duration of alerts in milliseconds
#define SetAlertMs  1500
#define LineAlertMs  400
#define GapAlertMs   200
#define IntvAlertMs 3000
//
// 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
```