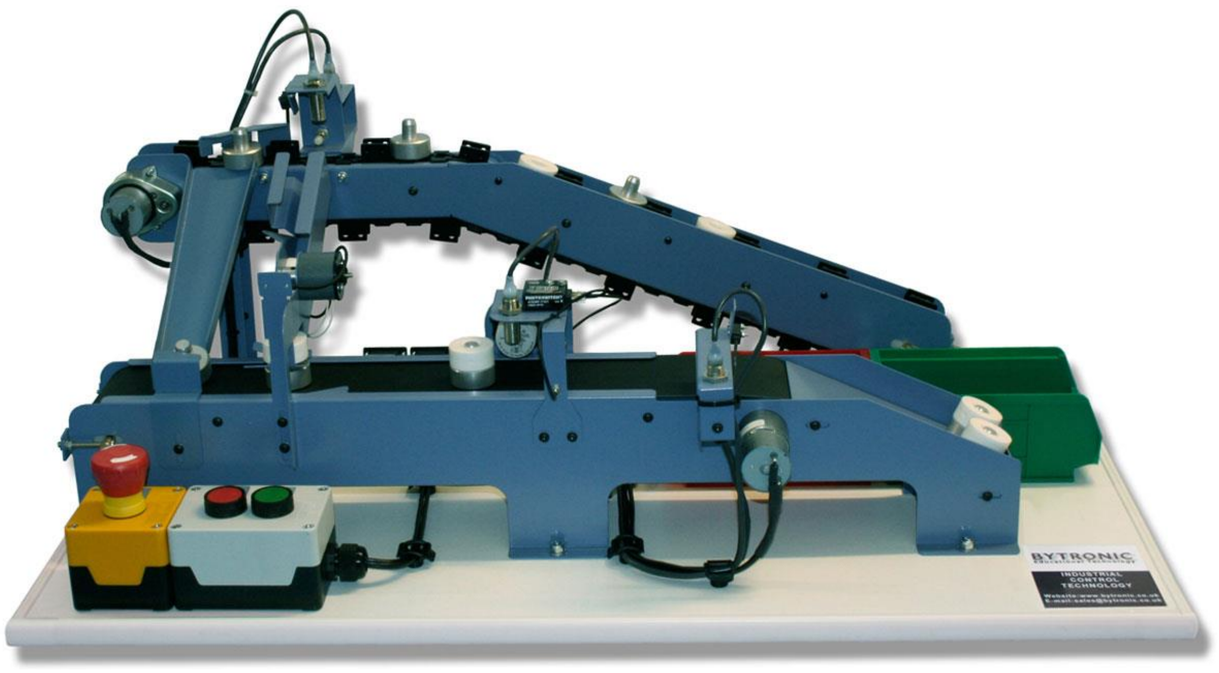
MSE 310 Project:
Industrial Control Trainer 3



Group: TH5

Ashley Powell-301293252

Parshant Bombhi- 301255126

Ataur Rehman- 301253848

INTRODUCTION TO ELECTRO-MECHANICAL SENSORS AND ACTUATORS
MECHATRONIC SYSTEMS ENGINEERING

Contents

Table of Figure

Introduction

For this project, we need to operate the assembled Industrial Control Technology (ICT) setup. This setup comprises of sorting area, assembly chute, sensing station and reject area. When all the widgets are loaded onto the top belt, the sorting area separates the metal pegs and plastic rings. The plastic pegs are then sent to the assembly chute. The assembly chute is responsible to dispose a plastic ring every time a metal peg passes through on the bottom belt. Later, the sensing station is used to detect whether the object passing through the infra-red fiber optic through beam sensor is an assembled piece or not. Based on the input from this sensing station, we can then actuate the rejecting solenoid to dispose off the unassembled pieces. Lastly, we are also required to have a means of system recovery in case of power loss.

Flowcharts

Reading Sensor Data

To successfully detect various forms of data, we are implementing either a rising edge detection or a falling edge detection. This ensures that the sensor data being used was either detected when an object is entering or leaving the sensing station. This method eliminates all the arbitrary sensor data and only writes to the file when changes occur. In other words, all the crucial tasks of reading and writing into a file are only done when change is detected and terminated otherwise.

Belt Functions

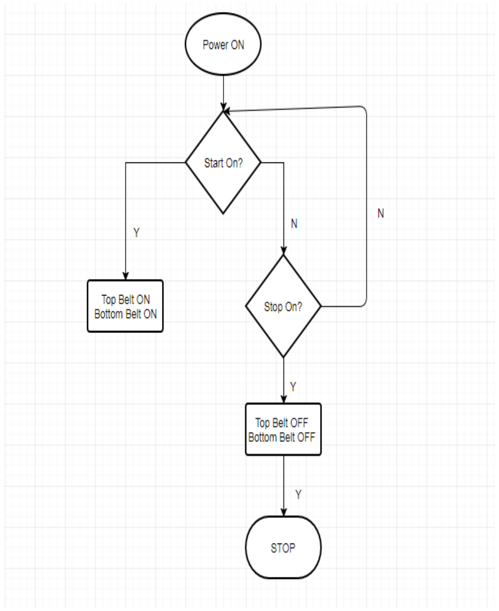


Figure 1 Belt Functions Flow Diagram

From the above figure, we can understand the simple logic that when the start is pressed, the top belt and the bottom belt, both turn on. These belts stop when the stop is pressed.

Sort Area

The challenge in programming the sorting area correctly is to read the sensor when changes happen. Since there is a time delay between the activation of IR sensor and the inductive proximity sensor, we decided to use a file “MetalPiece.txt” in which writes “1” when the inductive proximity sensor is activated. Later, when the IR sensor is activated, we read this “MetalPiece.txt”. If the IR sensor reads “1” then it qualifies as a metal piece and is the actuator (sorter solenoid) is not actuated. However, if the IR sensor reads nothing, then it qualifies as a plastic ring and sorter solenoid is actuated and sent towards the assembly chute. After every iteration of “while true” loop, it is crucial to reset the file and hence a “0” is written into “MetalPiece.txt” in a flat sequence. Furthermore, we would also have to check the hopper queue if the number of elements in the queue is 5. If the number of elements in queue is indeed 5, then we would not actuate the sorting solenoid. We will then reject this plastic ring at the last stage of this assembly. The overall algorithm can is shown in the figure below.

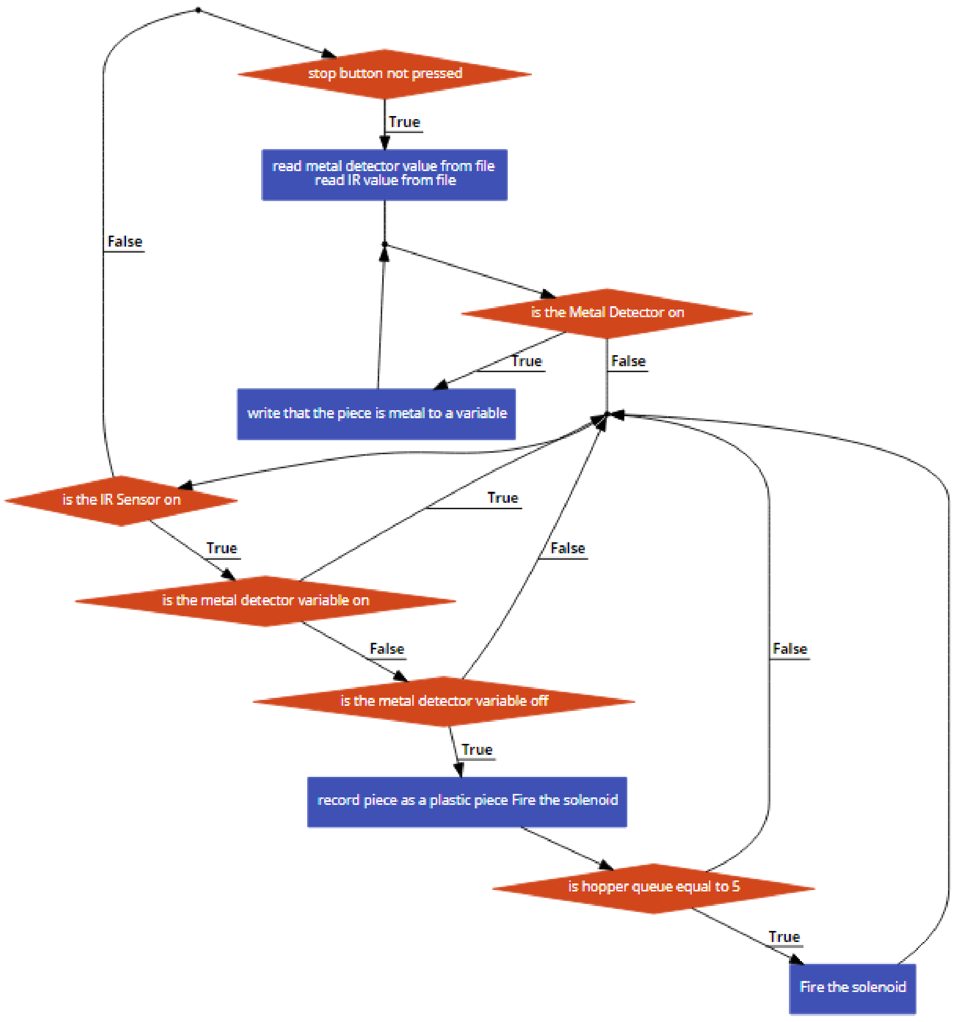

stop button not pressed
True
read metal detector value from file
read IR value from file
False
is the Metal Detector on
True
False
write that the piece is metal to a variable
is the IR Sensor on
True
True
False
is the metal detector variable on
False
False
is the metal detector variable off
True
record piece as a plastic piece Fire the solenoid
is hopper queue equal to 5
True
Fire the solenoid

Figure 2 Sorting Area Flow Chart

Assembly Chute

To correctly program the assembly chute, we need to implement two factors .i.e. firing condition and the hopper queue count. Following is the brief description of these factors

Firing Condition

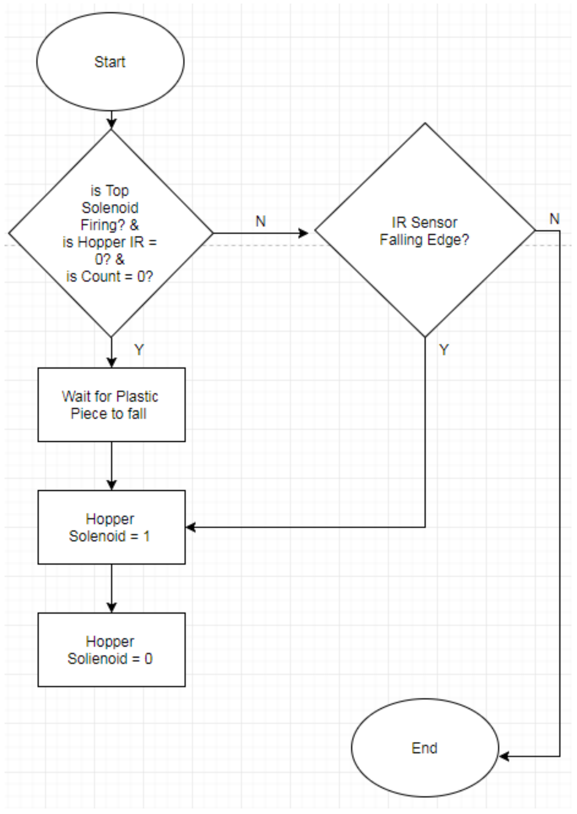

Start
is Top Solenoid Firing? & is Hopper IR = 0? & is Count = 0?
N
IR Sensor Falling Edge?
N
Y
Wait for Plastic Piece to fall
Y
Hopper Solenoid = 1
Hopper Soliengid = 0
End

Figure 3 Firing Condition of Rotary Solenoid

From the above figure, we can understand that the hopper would fire either when a plastic ring was taken by the metal peg or when the hopper was initially empty. If the hopper queue was initially empty, we first added a wait and then we actuate the hopper solenoid. The “wait” in actuating the hopper solenoid is crucial because we observed that if the hopper solenoid actuates at the same instance as the sorting solenoid, there is a possibility of hopper solenoid blocking the falling plastic ring. On the other hand, if the hopper queue was not empty, we first check if the metal peg took the plastic ring. Hence, when the Hopper IR sensor turns from true to false, we would actuate the hopper solenoid. Lastly, once the hopper solenoid is actuated we revert it back to its non-actuated state.

Hopper Queue Count

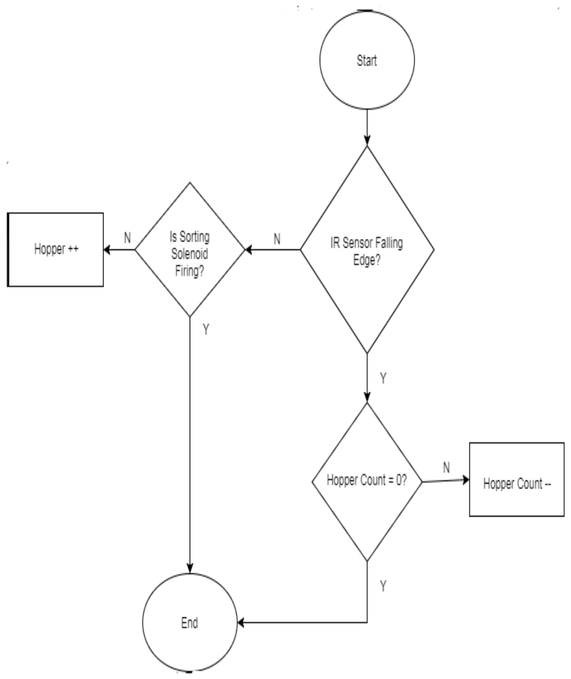


Figure 4 Hopper Count Flow Chart

Here, we used the inductive proximity sensor attached to the sorting solenoid. This sensor would indicate whether the sorting solenoid has fired. If the sorting solenoid has actuated, we wish to increment the hopper queue by 1. Additionally, if the Hopper IR changes from false to true (metal peg took the plastic ring), we wish to decrement by 1.

Sensing/Rejection Area

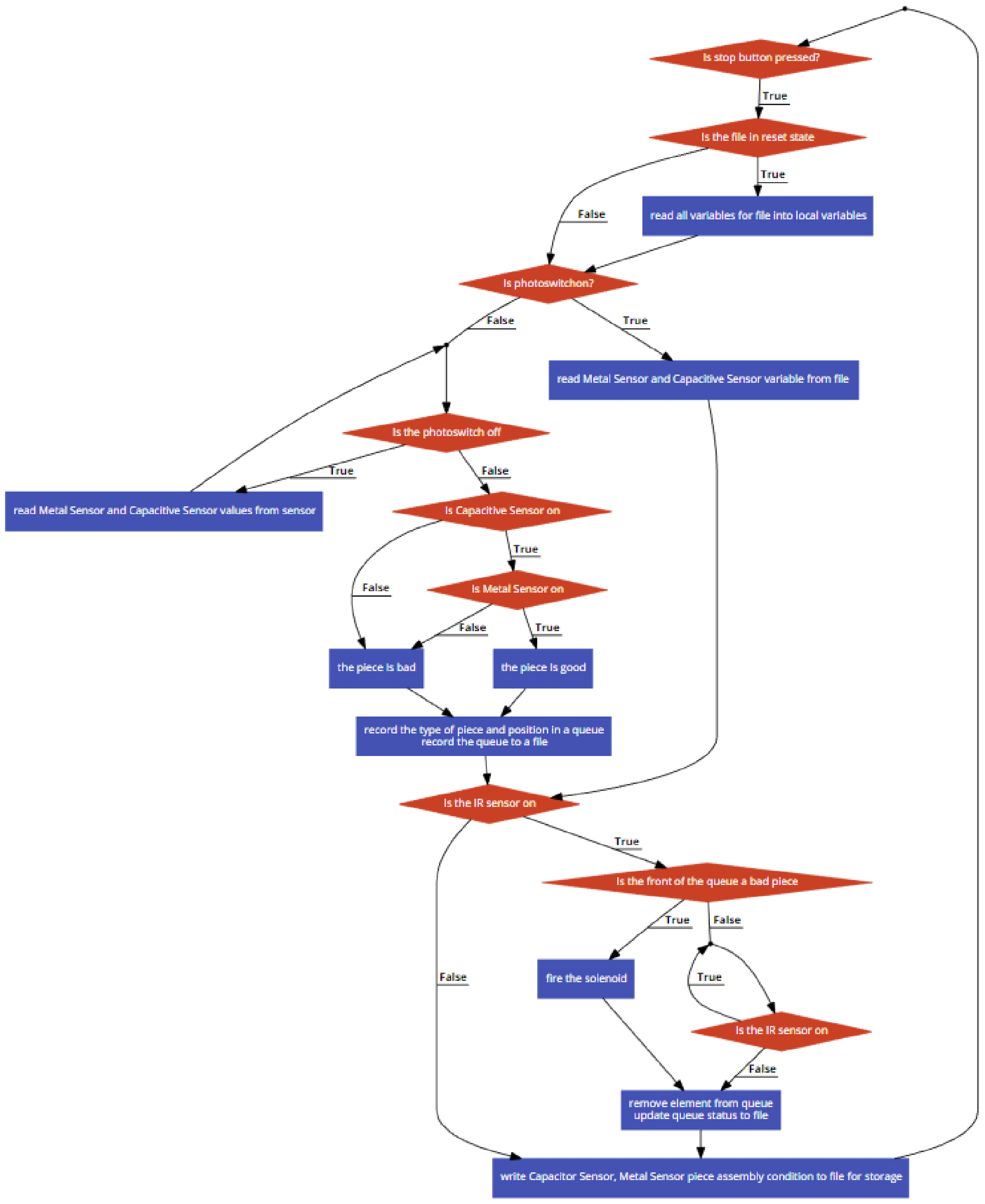

Is stop button pressed?
True
Is the file in reset state
True
False
read all variables for file into local variables
Is photoswitchon?
False
True
read Metal Sensor and Capacitive Sensor variable from file
Is the photoswitch off
True
False
read Metal Sensor and Capacitive Sensor values from sensor
Is Capacitive Sensor on
True
False
Is Metal Sensor on
False
True
the piece is bad
the piece is good
record the type of piece and position in a queue
record the queue to a file
Is the IR sensor on
True
False
Is the front of the queue a bad piece
True
False
fire the solenoid
True
Is the IR sensor on
False
remove element from queue
update queue status to file
write Capacitor Sensor, Metal Sensor piece assembly condition to file for storage

Figure 5 Sensing Flow Chart

Similar to the sorting area, we use the photo switch as a trigger to actuate the rejection area solenoid. The above figure shows the algorithm that we used to actuate the solenoid. While the Infra-red fibre-optic through beam sensor (through beam sensor) is on, we read the file where we are storing the values for capacitor and inductive proximity sensor for the rejection area. When the through beam sensor turns off, we need to check if the capacitor and inductive proximity sensor is on. Following is the table for the conditions.

Capacitor	Inductive Proximity	Condition
T	T	Good piece
F	F	Bad Piece
F	T	Bad Piece
T	F	Don’t Care

Figure 6 Reject Condition

Based on the above combinations, the pieces can either be qualified as a good piece or a bad piece. These pieces are then enqueued as they pass through the through beam sensor. Later, when the IR sensor near rejection solenoid turn true, we read the first element in the queue. Depending if the piece is a good piece (assembled) or a bad one, we actuate the rejection solenoid. Additionally, we dequeuer the piece after any piece is detected by the IR sensor at the rejection area.

Efficiency

We defined efficiency as follows:

$$\%\text{Efficiency} = \frac{number\ of\ assebled\ pieces}{number\ of\ total\ pieces} \times 100$$

To count the total number of pieces, we used the through beam sensor since it detects both objects metal and non-metal. If a part is assembled, the capacitor changes from true to false and adds this piece to a queue. It is at this time we increment the number of assembled pieces. Furthermore, an assembled piece contributes a count of two elements towards the total number of parts. If the capacitor does not turn on but the through beam turns on, we register it as a non-assembled piece and only increment the total number of pieces by 1. Again, we made use of file storage to ensure robustness and system recovery.

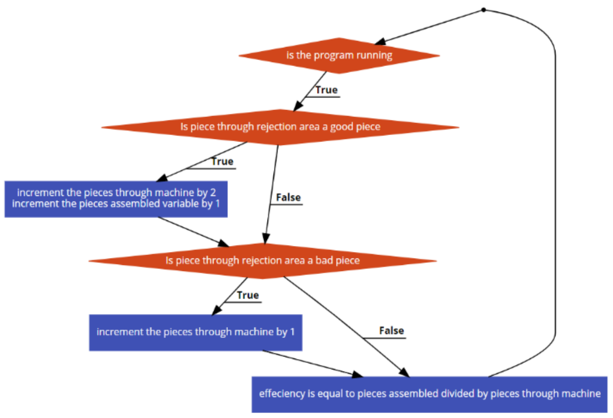

is the program running
True
Is piece through rejection area a good piece
True
increment the pieces through machine by 2
increment the pieces assembled variable by 1
False
Is piece through rejection area a bad piece
True
increment the pieces through machine by 1
False
effeciency is equal to pieces assembled divided by pieces through machine

Figure 7 Efficiency Flow Chart

The System Recovery

All the data that need to be recovered after system crash is constantly saved to a file when the data is being updated. At start of the file a prompt asks the user if they want to reset all the data. If they choose yes, then the counters for all the pertinent data are initialized as zero. If they choose to cancel, then all the data from the files is read and passed into the counters. This way, if for some reason there is a power loss and the user removed all the pieces, they can reset the system back to zero.

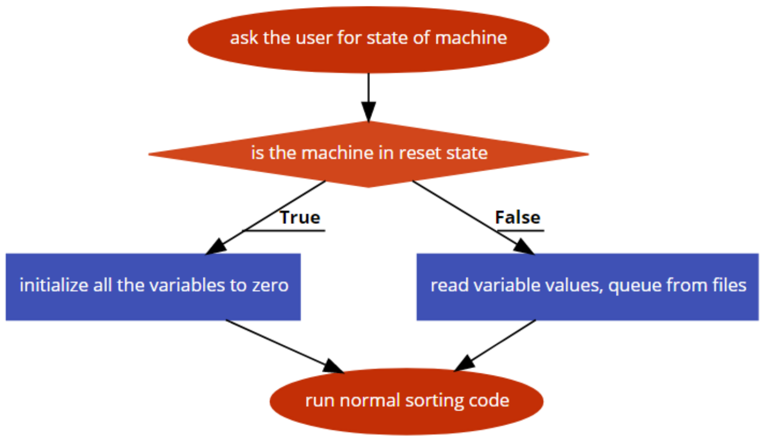

ask the user for state of machine
is the machine in reset state
True
False
initialize all the variables to zero
read variable values, queue from files
run normal sorting code

Figure 8 Reset Decision

Sensors Description

Inductive Proximity Sensor

The inductive sensors rely on a high-frequency magnetic field generated by a coil. When a metal object approaches the magnetic field an eddy current is induced in the surface. This current causes a damping in the oscillation of the magnetic field which can be detected by the sensor.

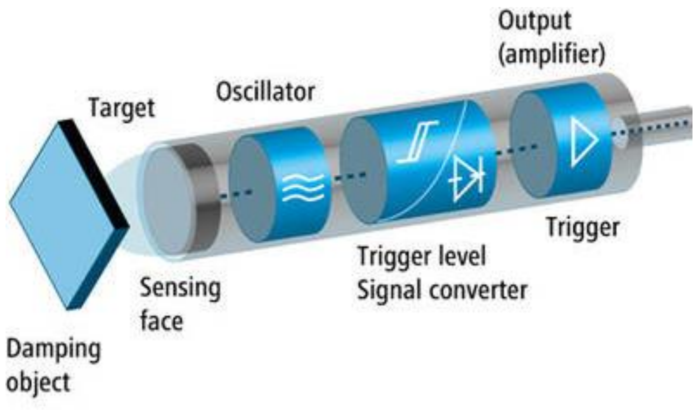


Figure 9 Inductive Sensor

Capacitive Proximity Sensor

Capacitive proximity sensors are similar to inductive sensors but instead of an electromagnetic field, it produces an electrostatic field. The sensor consists of metal plates which act as electrodes of an unwound capacitor. When an object comes into the path of the sensor, the capacitance between the electrodes changes dramatically. The change in oscillation increases the closer the object it to the sensor; This allows the user to set a threshold for when they want the sensor to have an output. Unfortunately, several ICT stations seemed like they had their threshold set to low because the sensor would often not detect an assembled piece.

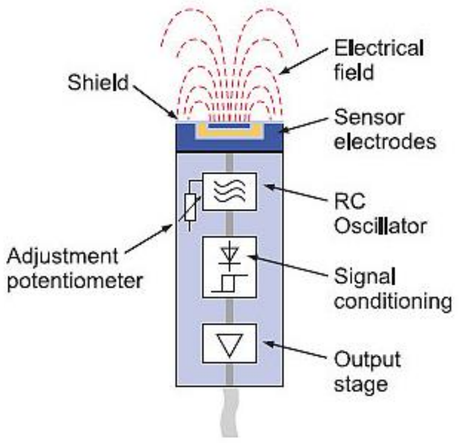


Figure 10 Capacitive Sensor

Infrared Sensor

An Infrared Sensor (IR sensor) is usually make up of two devices, a IR LED and a photodiode. The IR LED is orientated in such a way that the light bounces off the object and into the photodiode. The resistances and these output voltages of the photodiode circuit, changes in proportion to the magnitude of the IR light received. This allows the user to know when there is a piece in front of the IR sensor.

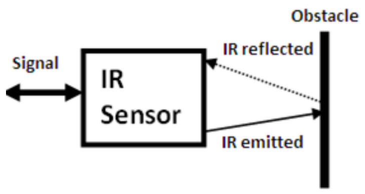


Figure 11 IR Sensor

A special version of an infrared sensor is the through beam sensor which is characterised by its two separate components: a transmitter and a receiver. A through beam sensor has its IR light directed towards a receiver by a fibre optic cable which allows it to have a very long range.

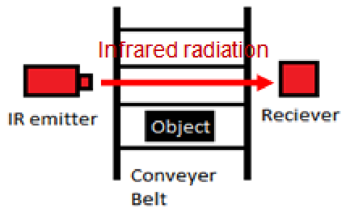


Figure 12 Through Beam IR Sensor

Actuators Description

Solenoid

There were two types of solenoids used in this project. One was the linear solenoid and the other the rotatory solenoid. Both work on the same principles but have different degrees of motion.

A solenoid is a coil of wire wrapped around a ferromagnetic core. When current is passes through the coil a magnetic field is created that accelerates the core in a certain direction. Solenoids were used in a large extent to both sort, assemble and reject all the pieces to be assembled. The linear solenoid relies on a regular spring to return its core back to its starting position, while the rotary solenoid uses a coil spring.

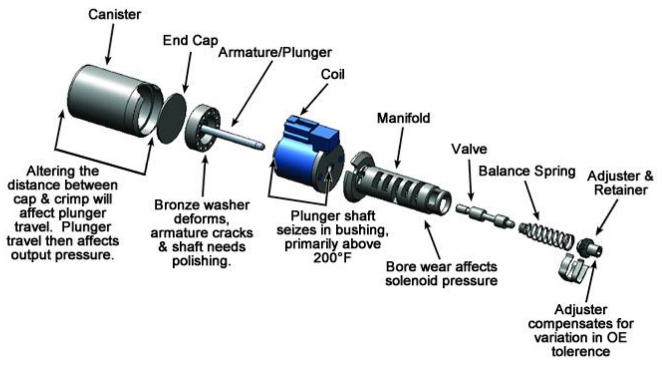

Canister
End Cap
Armature/Plunger
Coil
Manifold
Valve
Balance Spring
Adjuster & Retainer
Altering the distance between cap & crimp will affect plunger travel. Plunger travel then affects output pressure.
Bronze washer deforms, armature cracks & shaft needs polishing.
Plunger shaft seizes in bushing, primarily above 200°F
Bore wear affects solenoid pressure
Adjuster compensates for variation in OE tolerence

Figure 13 Linear Solenoid

Motors

Like the solenoids, a DC motor work on current following through coils. The current produces a magnetic field which causes a rotor, with permanent magnets attached to it, inside the center of the coil to rotate. These DC motors were simply used to move the conveyor belts across the station.

Summary

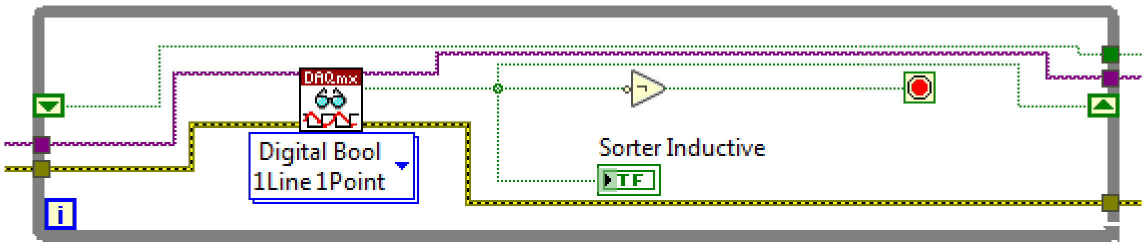
This project helped us to makes a robust program for an assembly line. Even though it was on a small scale, we learnt about the problems encountered while using sensors and actuator. Furthermore, we learnt about trigger mechanisms and the logic behind them to fire an actuator for complex conditions. We learnt to design a sorter from the sorting area. The challenge here was to correlate sensors even when there is a time delay between inductive proximity sensor and the IR sensor. Later, as we moved onto the assembly chute, we first learnt to make practical use of a queue. When we programmed the sensing station, we encounter a bigger time delay between the capacitive proximity sensor and the through beam sensor. To successfully implement this, we had to use a storage system (we used a file system). Later when we reached the rejection area, we needed to read the stored values. The option of having a reset button was would recommend of having a flat sequence and separate initialization process for variables and files. These steps together contributed to a robust program for the ICT setup. Lastly the usage of file system was crucial for system recovery. Overall, there was a lot to learn in this project and we executed our knowledge efficiently and successfully

Discussion

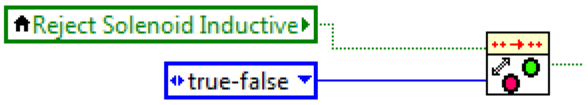
While programming the ICT set up we encountered many challenges. Firstly, for the sorting area, we realized that there was a time delay between the inductive proximity sensor and the IR sensor. To program this area correctly, we made use of imbedded while loops and case structure which was later replaced by a simple read and write file system when change in sensor data is encountered. Secondly, for the assembly chute, we encountered that the IR sensor at the assembly chute would sometimes be triggered twice if the plastic ring would not perfectly be assembled. To solve this, we had to add a time interval between the piece to be dispensed and for it to settle down. Later, at the sensing station, we noticed the capacitive proximity sensor would only turn on when the assembled piece is touching the capacitive proximity sensor. This was a design problem and hence we had to constantly check if the sensor turned on. Lastly, since we were using files to write sensor data at sensing station and read at the rejection station, we encountered a problem where a file can either be written or read at an instance and Is impossible to read and write into a file at the same time. To solve this issue, we decided to use queues where we enqueued at the sensing station and dequeued at the rejection area. At the rejection area, we would only read the first element in the queue and then depending whether it was a 1 or a 0 (assembled or non-assembled) we would actuate the rejection solenoid. The use of queues and file systems later made our program efficient for system recovery as well as resetting the ICT setup. In conclusion, even though we encountered challenges for this project, we can solve them and performing exceptionally at the demo.

LabVIEW Programming

While programming the ICT, a steep learning curve occurs this is caused by the user thinking treating LabVIEW as sequential programing where one action happens another. However, in LabVIEW all sensors are reading all the time and decisions are made simultaneously. Therefore to program the machine to perform a task only when the falling or rising edge of a signal are seen either while loops or edge trigger can be used.

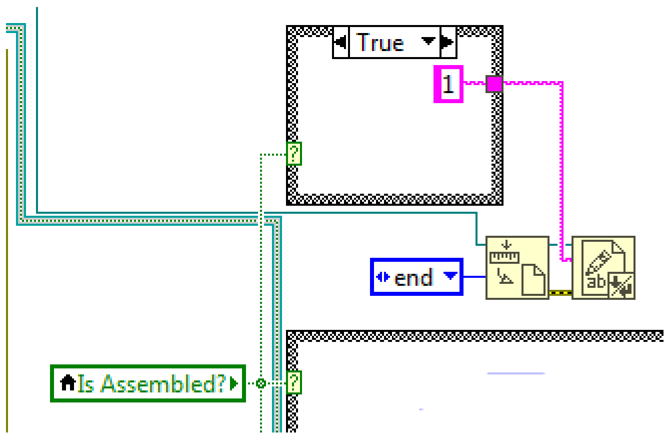


However, using this method is can be wasteful of memory can be prone to user error. Therefore, a block implementation as shown below performs the same actions and allows the user to the edge on which the action should be performed:

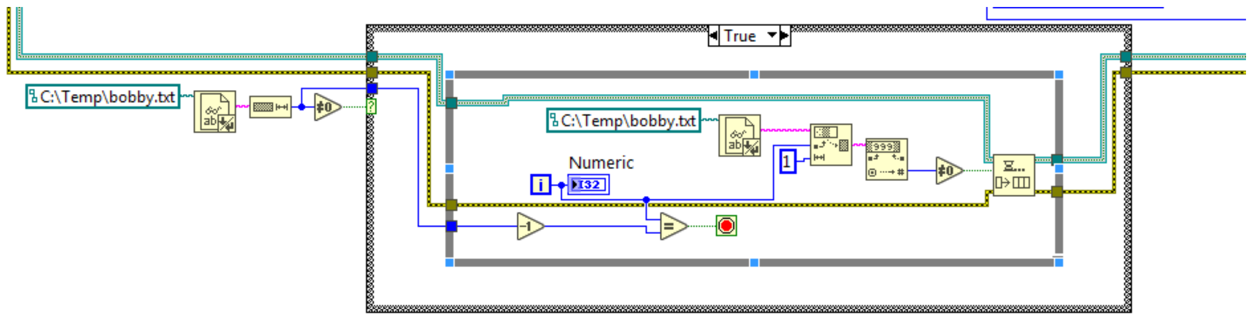


Queue File Loading

One of the toughest challenges in the project was to create a file which would hold the reject area queue in real time. Firstly, each time an element was added to the queue it would also be saved to the end of a file, as shown below.



Adding an additional queue to a file is rather easy, even more, dequeuing an element is rather trivial as mainly the file is read with an offset of 1 and then wrote back into the file. Doing so removes the element in the front of the queue. Finally, during start-up if the user choices to remember the previous the file is opened and checked to see if it is an empty file. If the file has content than each string in the file is read in a while loop controlled by the length of that code. Finally, the element is enqueued and ready for normal.



The enqueuing code is the most complex memory function we had to implement.