FROBOTICS
TEAM 4150

Control System Training

MODULE 7 – Finite State Machines

Copyright Notice

Finite State Machine - Definition

Definitions:

- **Finite State Machine** - A computational model consisting of a predefined number of states. Only one state can be active at a time. The active state performs some action, which could be nothing, then decides which state, including itself, becomes the next active state.
  - States cannot be dynamically added or removed
  - One state is defined as the starting state.
  - A state machine may have an end (exit) state or may be defined to execute perpetually.
  - Can be used to process Sequential Boolean Logic
  - There are many other uses such as language processing, and compilers.

How the machine works

- **The machine:**
  - Executes code for the initial state
  - Loops forever executing the code for whatever state is designated to be next, or until a state requests it to stop
- **Recommendation for FRC robots:**
  - During each robot control system “scan” cycle, 20 milliseconds, a single “state” is executed.
  - In some cases, this will mean that a 20 millisecond delay will be performed to get to the next state. Generally, 20 milliseconds is too small to be concerned with. If it is, then the state design should be reconsidered to avoid this delay.

What each "State" Does

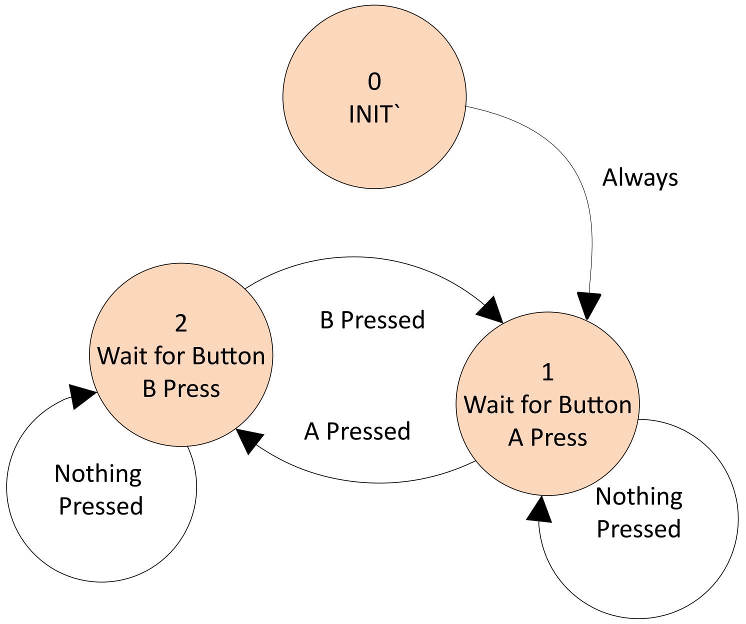
- **Each state:**

  1) Performs some computation, or initiates some action. The action could be:
     - Starting or stopping a motor
     - Opening or closing a solenoid
     - Setting a variable for another part of the code to use
     - A combination of things

     Doing nothing is a valid action.

  2) Decides which state should execute next. The next state could be:
     - The same state
     - A known state determined during design
     - One of several states determined by logic

Drawing State Machines



Actions:

----

0 – INIT
    None
1 – Wait for button A press
    Turn off light
2 –Wait for button B press
    Turn on light

- **Each State is a circle. Define each state with a unique name and/or state number.**
- **Lines between states are transitions. Document the conditional causing each state transition.**
- **Document actions taken by each state. This can be done in the circle defining the state. However, a separate list allows for more detailed description of the actions.**

Defining States

- **Define a separate initialization state – State 0**
  - If somehow there is an error, and the state is set to zero, then this will cause initialization.
- **Include an error recovery state**
  - Recycle system, wait, and allow system to try again.
- **Keep states simple**
- **Keep states simple**
- **If possible, join common sequences of actions together to reduce the number of states**
- **Actions of states should not need to include conditionals.**

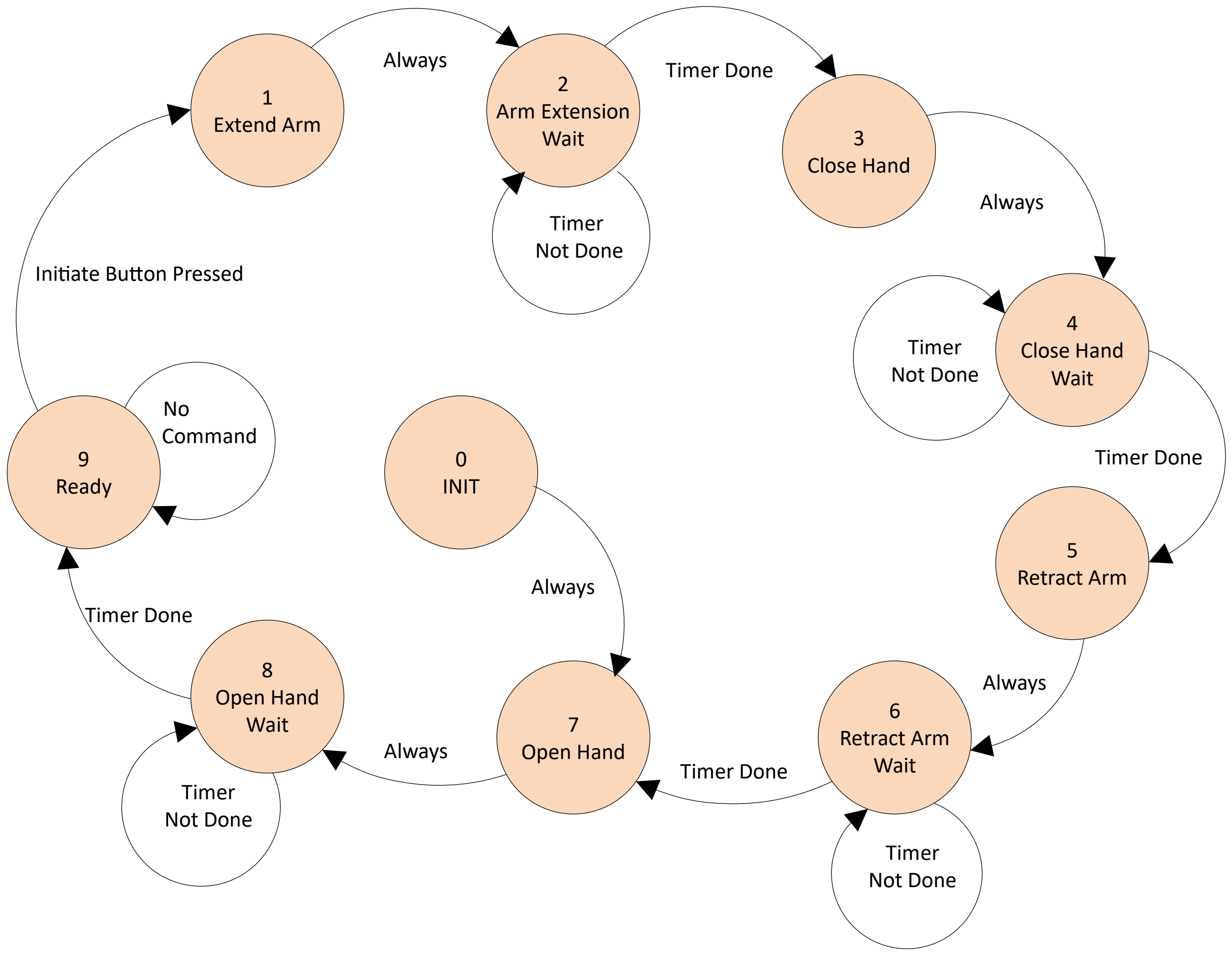
Designing Finite State Machine - 1/3

- **Sample Problem – Cube Capture**
  - Robot has a "hand" with positions:
    - Opened = false
    - Closed = true
  - Robot has an "arm" with positions:
    - Retracted = false
    - Extended = true
  - System is "ready" when "hand" is opened and "arm" is retracted
  - User pushes button to initiate "capture cube". Auto repeat of cube capture is not allowed. Ensure user pushed button for 60 milliseconds.
  - Sequence of cube capture is:
    - Close "arm" extension solenoid. Wait 1 second for arm to extend.
    - Close "hand" solenoid. Wait 2.0 seconds for "hand" to settle.
    - Open "arm" extension solenoid. Wait 1.3 seconds for arm to retract.
    - Open "hand" solenoid to release potential cube into bin. Wait 1.5 seconds for cube to drop before allowing next "grab" action.

Designing Finite State Machine 2/3

- **List:**
  - Inputs
  - Outputs
  - Individual actions, and sequence of actions
- **Optionally, draw a Logic Diagram Graph similar to sequential logic.**
- **Draw the state diagram:**
  - Start with an “initialization” state
  - Add states waiting for events that start sequences of actions – sensor inputs, user button press, autonomous request.
  - Add states for sequences of actions. Often this occur in pairs – 1) take action, start timer, 2) wait for timer to expire before selecting next action.
- **Define for each state:**
  - Actions performed, if any
  - Tests used to determine next state

Designing Finite State Machine 3/3



Actions:

----

0 – INIT
- Open Hand
- Retract Arm

1 – Extend Arm
- Extend Arm
- Initiate 1.0 Second Timer

2 – Arm Extension Wait
- None

3 – Close Hand
- Close Hand
- Initiate 2.0 second timer

4 – Close Hand Wait
- None

5 – Retract Arm
- Retract Arm
- Initiate 1.3 second timer

6 – Retract Arm Wait
- None

7 – Open Hand
- Open Hand
- Initiate 1.5 second timer

8 – Open Hand Wait
- None

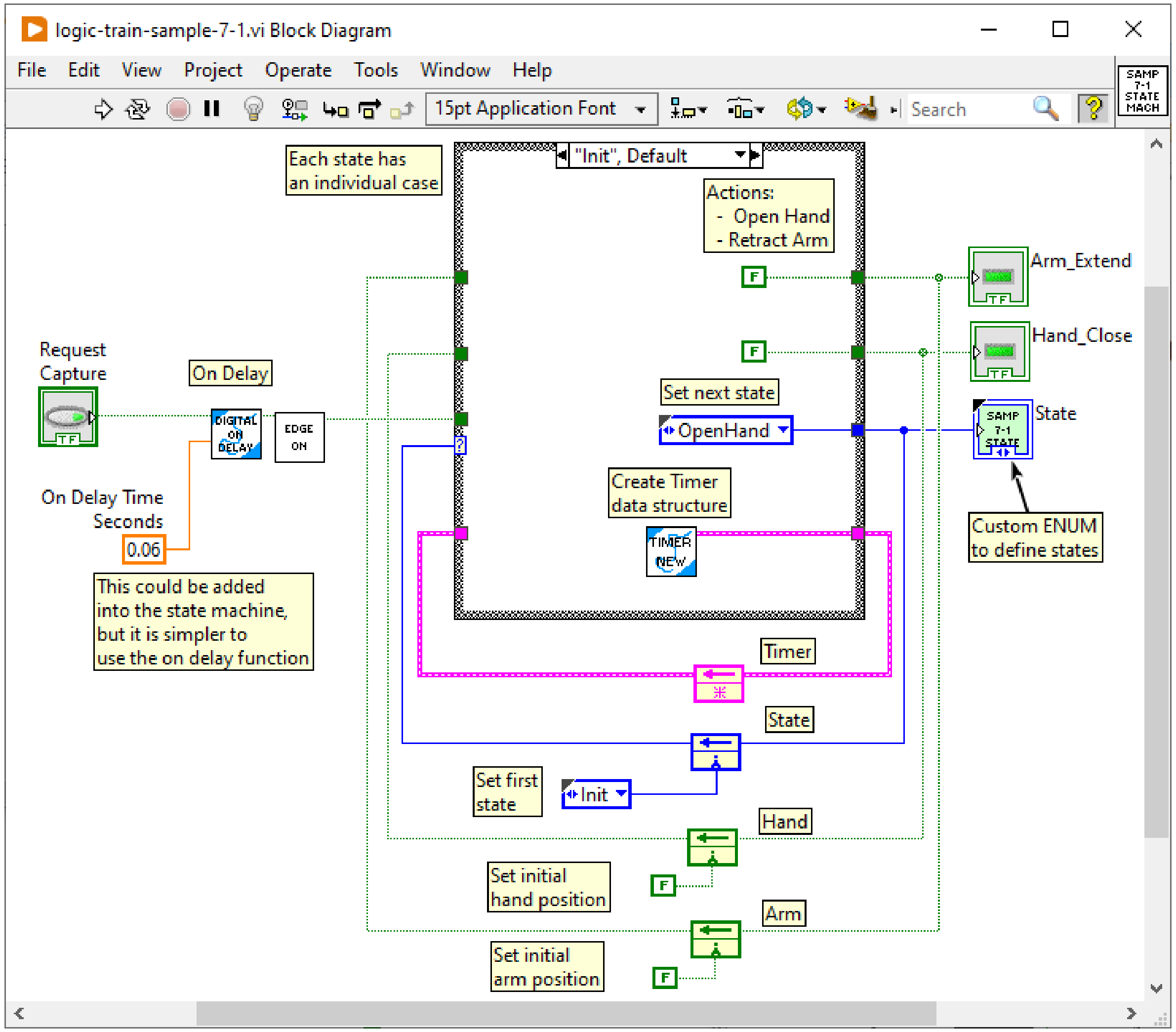
9 – Ready
- None

Notes:

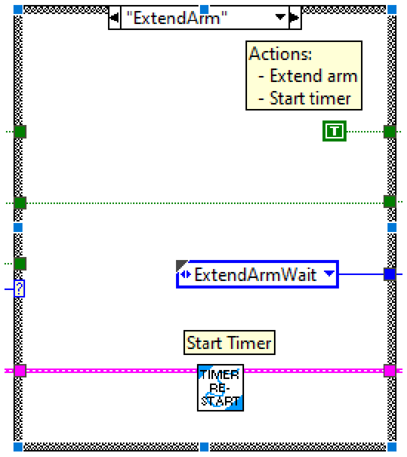
----

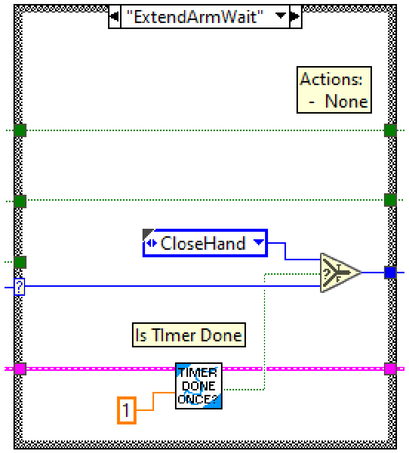
1) Initiate button on-delay, permissive checking, and edge triggering done outside of finite state machine. (This could be done either way.)
2) Cancel button would transition to 0-INIT state.

Designing Finite State Machine - LabVIEW


logic-train-sample-7-1.vi Block Diagram
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
"Init", Default
Each state has an individual case
Actions:
- Open Hand
- Retract Arm
Arm_Extend
Hand_Close
Request Capture
On Delay
Set next state
OpenHand
State
On Delay Time Seconds
0.06
Create Timer data structure
Custom ENUM to define states
This could be added into the state machine, but it is simpler to use the on delay function
Timer
State
Set first state
Init
Hand
Set initial hand position
Arm
Set initial arm position

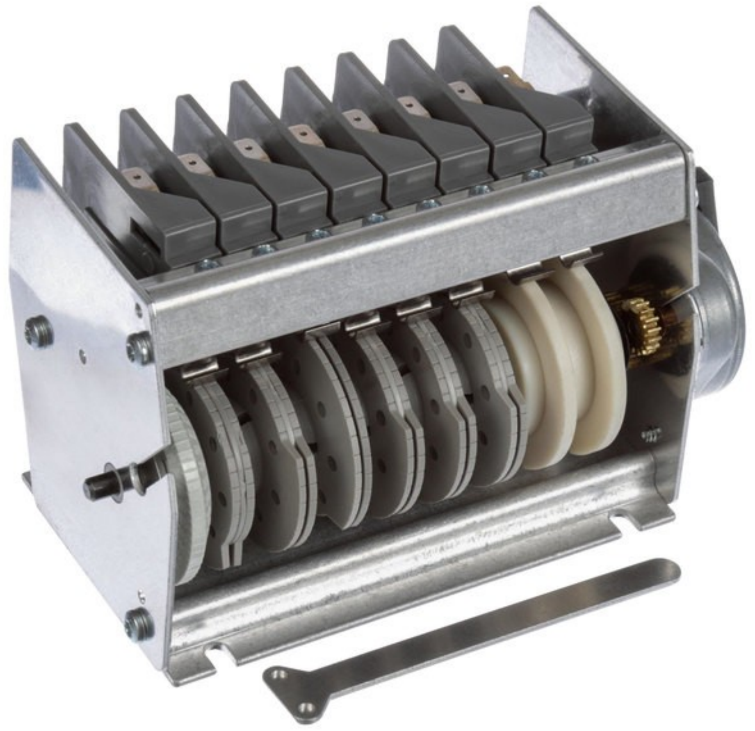
Other cases: perform action and start timer; wait for action to complete.
Other pairs of states are similar



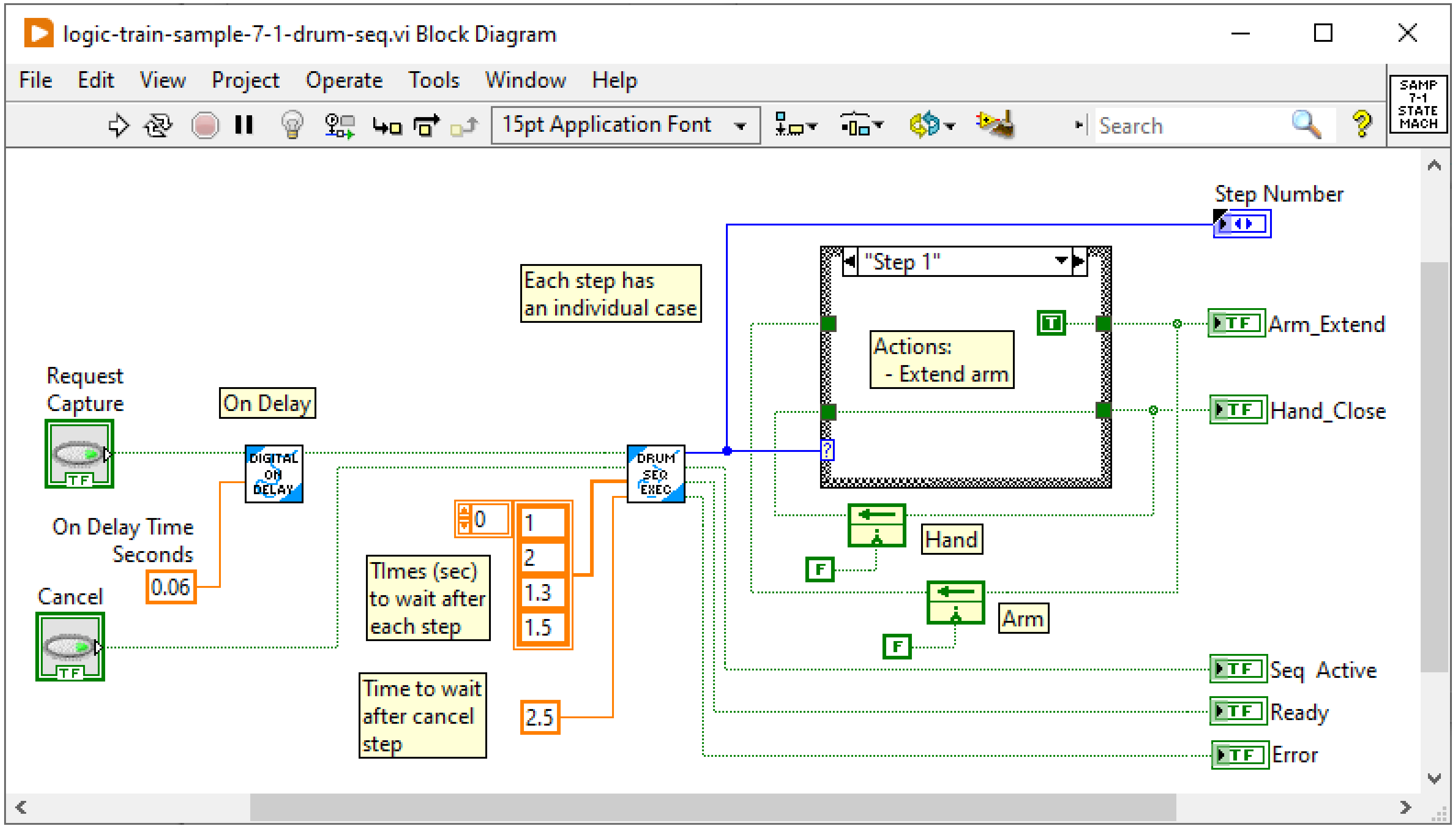


Drum Sequencer

- **Drum Sequencer – A simplified state machine that performs a "circular" set of actions such as – Wait for Command, Action 1, Wait 1, Action 2, Wait 2, Action 3, Wait 3, back to Wait for Command.**
- **Named after the motorized rotating mechanical drums with sets of contacts.**
- **Eliminates the need to deal directly with timer code.**



Drum Sequencer - LabVIEW



Other cases perform other actions. There are special cases for: Initialize, Cancel, Error. There is a "Do Nothing" case while the post step waits occur. This case is expected to not change any outputs.

Sequential Logic or State Machine

- **Many problems can be solved either by sequential logic or state machines. How to choose between them:**
  - Simple machines are probably implemented more easily using sequential logic.
  - More complex machines are probably easier to implement using a finite state machine.

Exercise 7.1 – Shoot Flying Disc

- **Flying Disc shooter system has two outputs:**
  - Shooting motor (on/off)
  - Solenoid to push flying disc into shooter wheel.
- **Flying Disc shooter system has one sensor:**
  - Limit switch indicating shooting system contains a flying disc.
- **User pushes a button to shoot flying disc. Ensure user meant to push button. Button must be pressed for 0.060 seconds before initiating action. (Robot cycle time is 0.020 seconds).**
- **Only shoot a flying disc if system contains a disc. Also battery voltage must be > 11.5 volts. Only one disc can be shot at a time.**
- **Shooting motor takes 3 seconds to spin up to speed. Engage solenoid for 2 second to push flying disc into shooting wheel. Allow 2 more seconds for shooting to complete. After shooting is complete, stop motor. (For now, don't allow continuous shooting.) It takes 5 seconds after shooting for the next flying disc to be in place ready to shoot.**
- **Allow user to press a Cancel button. The cancel button must be pressed for at least 0.060 seconds before becoming active. After the Cancel, force a 5 second reset before allowing a new shot.**
- **Design shooting logic. Also provide "ready to shoot" digital for dashboard display. Use a finite state machine and perhaps combinatorial logic from module 4.**

Robot Programming 03

- **Complete Robot Programming Training 03**

Exercise 7.2 – Shoot Flying Disc Robot Code

- **Implement the solution to 7.1 on a robot.**
- **The limit switch inputs use:**
  - Shooting system contains a flying disc – DIO 0
  - Battery voltage > 11.5 volts – DIO 1 (Alternatively find a library VI that provides battery voltage.
  - Shoot button – DIO 2   (Alternatively use a joystick button.)
- **The outputs are:**
  - Run motor – Relay 0 – (Forward only)
  - Shoot disc cylinder – dual solenoid –
    PCM 0, channel 0 – shoot disc, channel 1 – normal position
- **Use the robot project "put-name-here". The only VIs that need to be modified are in the "FlyingDiscShoot" sub-directory. They are:**
  - FlyingDiscShoot_Open - One time initialization goes here
  - FlyingDiscShoot_Execute - Code to periodically execute goes here