# Lab 06: Obstacle Course – Tutorial

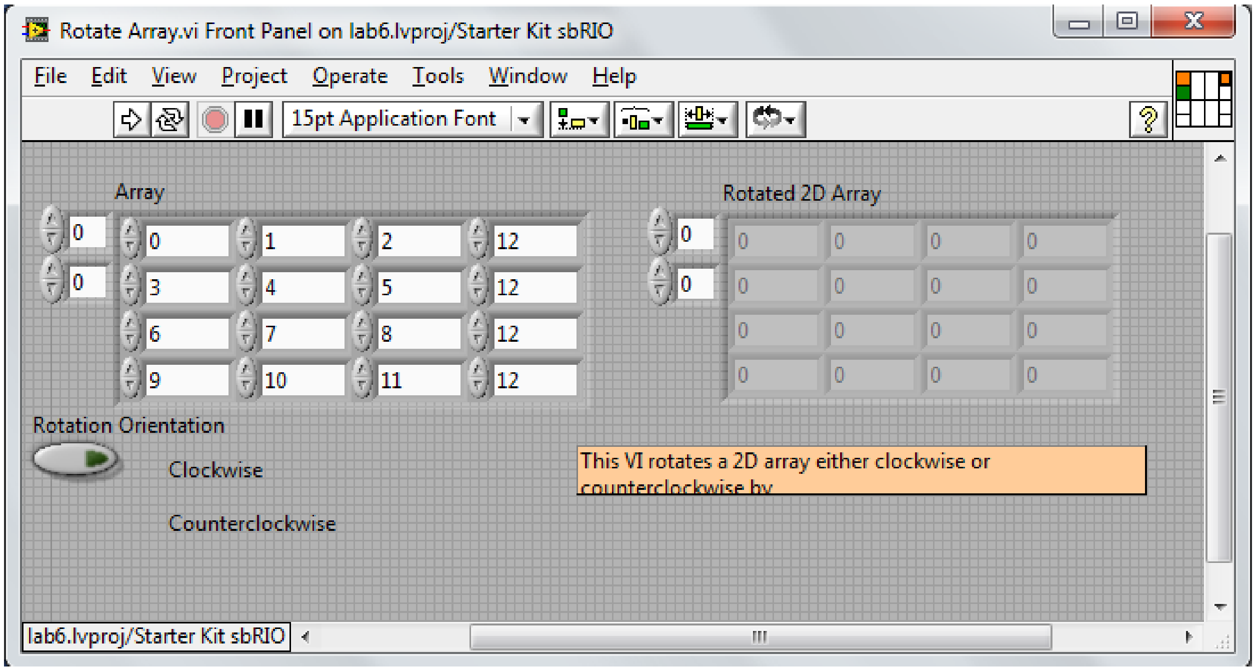
Note: The idea behind this solution is to remove the requirement of the robot to always face forward after each movement in the map. To do this, the map will be rotated each time the robot turns in order for it to always be facing forward in the map. Also, a marker will be placed before the array is rotated in order to obtain the robot's position after the 2D array is rotated. The marker will then be located in the array and set as the new row and col position.

**Step 1**: This solution focuses on implementing A* path planning to navigate the obstacle. However, due to the ramp obstacle, the code developed in the tutorial for Lab 05 will not allow the robot to navigate the course successfully because of the robot constantly readjusting itself to always face forward after each step taken along the path in the map. Start LabVIEW(LV) Robotics 2009, and follow the steps in Lab 05 tutorial to set up the code for A* path planning. Instead of main VI being called "Implementation", name the main VI as "Obstacle Course." All of the other subVIs may retain the same name from the previous tutorial.

**Step 2**: Only a few changes will be made to the main VI that calls the other subVIs and several changes will be made to the "Determining Direction" subVI. A new subVI will be used called "Rotate Array" in this subVI. We will first configure this subVI to be used with our code.

**Step 3**: The map array must be rotated and we have already obtained code to rotate 2D arrays but now we must take out some of its controls and replace some controls with constants.

**Step 4**: The first thing to do is to change the control that sets the degree at which the array is rotated. For our code, only a 90 degree turn is needed so change that control to a constant and make sure that it is set to 90. We need the counterclockwise or clockwise Boolean control for when the robot turns left or right. Lastly, make sure the connector pane is the same as all the rest of our subVIs and that the connections have been made. The "Rotate Array" subVI should end up looking like this:


Rotate Array.vi Front Panel on lab6.lvproj/Starter Kit sbRIO
File Edit View Project Operate Tools Window Help
15pt Application Font
lab6.lvproj/Starter Kit sbRIO

Array

Index: 0, 0

| | | | |
|---|---|---|---|
| 0 | 1 | 2 | 12 |
| 3 | 4 | 5 | 12 |
| 6 | 7 | 8 | 12 |
| 9 | 10 | 11 | 12 |

Rotated 2D Array

Index: 0, 0

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

Rotation Orientation

Clockwise

Counterclockwise

This VI rotates a 2D array either clockwise or counterclockwise by

Fig. 6-1

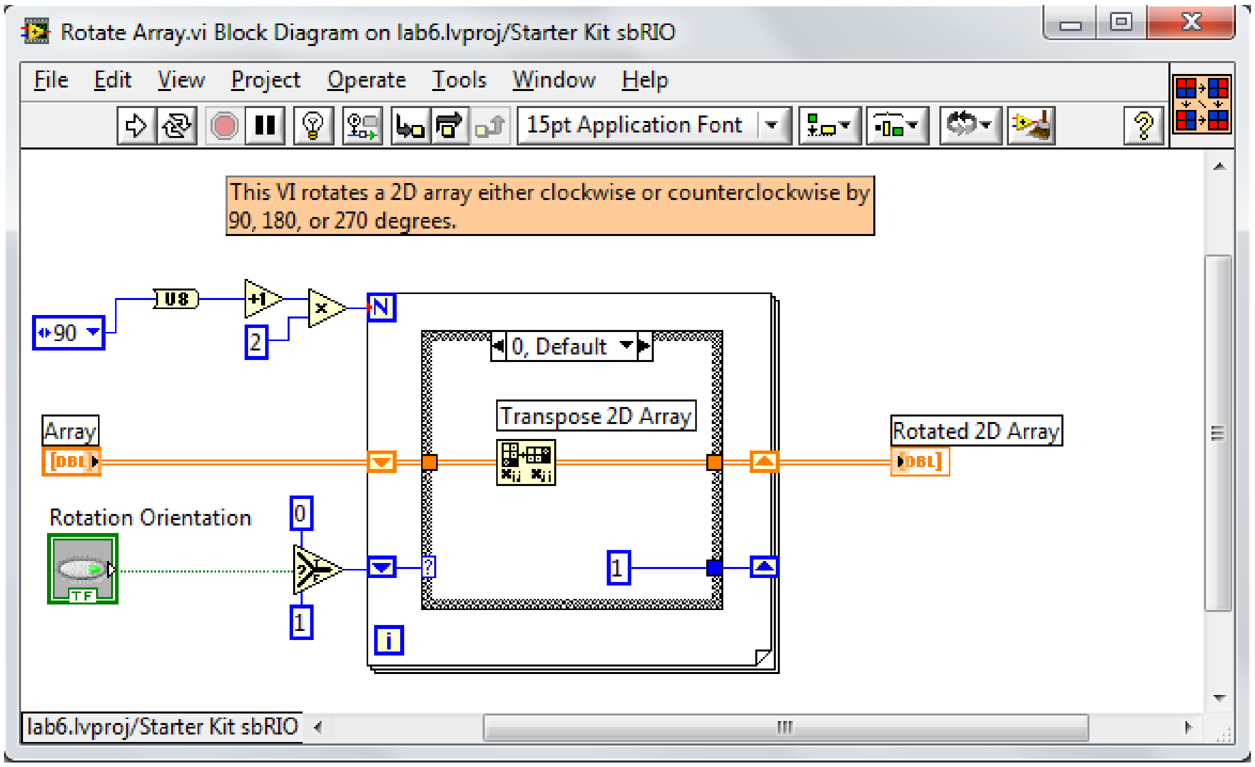

Rotate Array.vi Block Diagram on lab6.lvproj/Starter Kit sbRIO
File Edit View Project Operate Tools Window Help
15pt Application Font
This VI rotates a 2D array either clockwise or counterclockwise by 90, 180, or 270 degrees.
90
U8
+1
2
N
0, Default
Transpose 2D Array
Array
DBL
Rotated 2D Array
Rotation Orientation
0
1
1
i
lab6.lvproj/Starter Kit sbRIO

Fig. 6-2

**Step 5**: Now to make a slight change to the main VI. Instead of the current position being changed to a value of 1 after the robot has moved to before the robot moves in case the array is rotated and the current robot's position changes to a different row and col value. This will not affect the operation of checking around the robot's position to determine how the robot will move. To accomplish this, go to the block of the flat sequence that takes the number of blocks in the path and passes it to a for loop that has a flat sequence inside of it and switch the first two blocks in the flat sequence. That way the value of the current position is set to 1 and then in the next block the subVI "Determining Direction" is called.

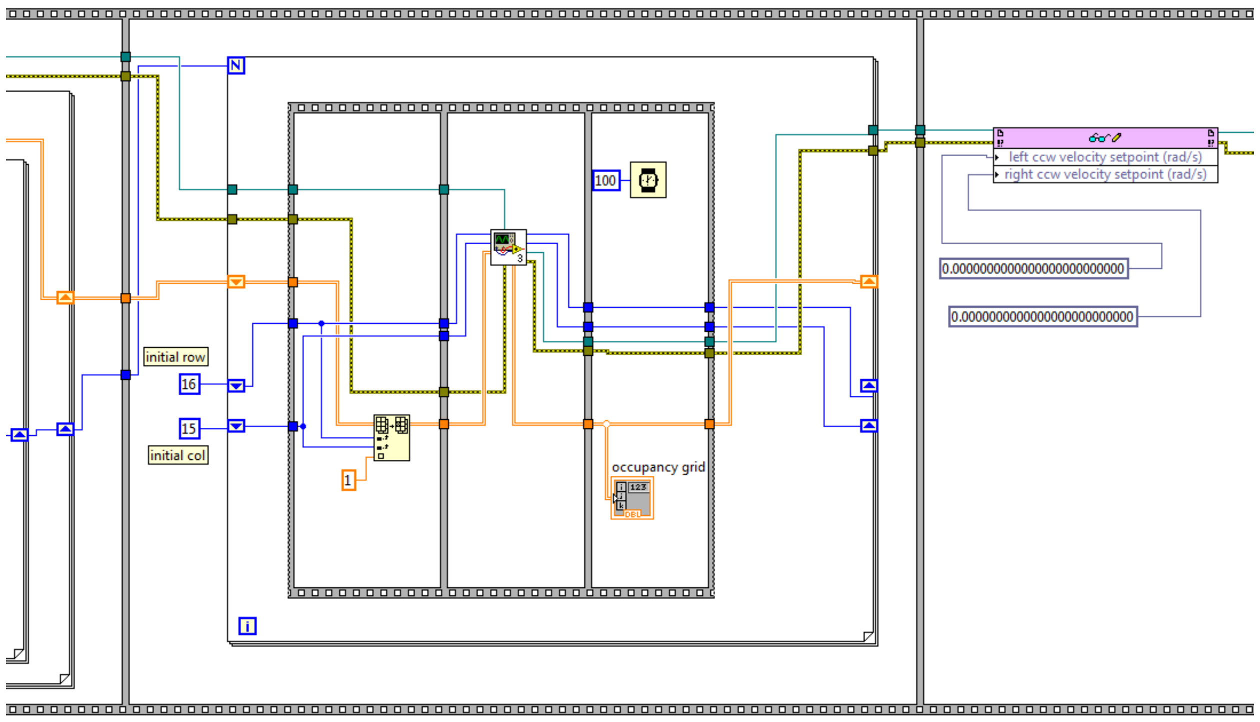


Fig. 6-3

**Step 6**: Now for the changes to the “Determining Direction” subVI. Since we will have the map following the turns that the robot will make, the robot will always be moving forward in the map. Thus, the case statements checking behind the robot, back diagonal right position, and back diagonal left position are no longer needed. The case statement for when the robot must move straight forward will remain the same. Therefore, the case statements for checking the right position and left position must be changed. Also, the robot cannot move in a diagonal direction in the obstacle course because of the ramp and the walls around each turn in the course. Thus, the case statements for checking front diagonal right and front diagonal left must be changed as well.

**Step 7**: To start off we will change the case statement for checking the right position. We will need to wire the array into the case statements because we will need it later. In the flat sequence for this case statement, find the block in where you are turning the robot back 90 degrees to face forward again. Delete this block because we are now keeping the robot facing the direction it has turned and we no longer want it to always face forward.

**Step 8**: Next, we want to use the updated row and col values to place the marker at the position where the robot is moving to. Add a frame to the end of the flat sequence and place a “Replace Array Subset” subVI in the new block which can be located under programming and then navigating to the array functions. The value to be set can be any number that is not 1, -2, or 100. For this solution, a value of 5 was set to the input to this subVI in order to change the value in the updated robot’s position to a 5.

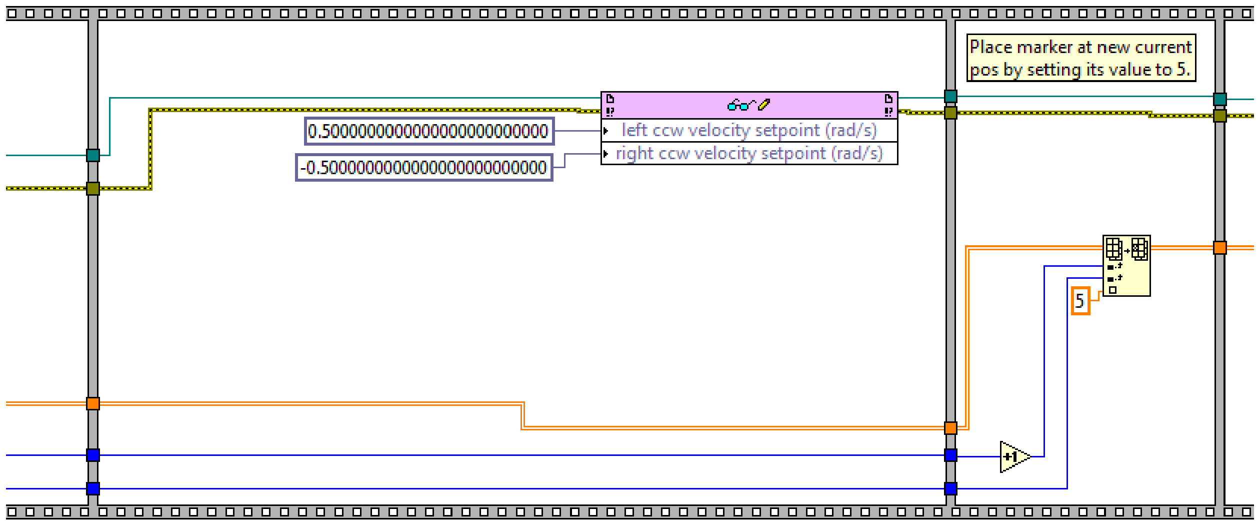

Place marker at new current pos by setting its value to 5.
0.5000000000000000000000000
-0.5000000000000000000000000
left ccw velocity setpoint (rad/s)
right ccw velocity setpoint (rad/s)
+1
5

Fig. 6-4

Note: In the figure above, the velocities are set to .5 and -.5 because after the robot has moved to its new position, we want the robot to stop moving. Thus, the last velocity the robot will see is either a very small velocity or zero. In this case, a very small velocity was used to test whether it will work or not to halt the robot. It turns out that this or a value of zero will do. If this does not stop the robot from constantly moving, then place a wait statement to keep the value at zero for some time. This will ensure the robot will not move.

**Step 9**: Now that we have our marker, it is time to rotate the array to the right to keep the robot facing forward in the map. Therefore, add a frame to the end of the sequence and place the subVI “Rotate Array” in the new block. Wire in the array input from the previous block. Wire a constant false into the Boolean input. A false constant will make the subVI rotate the array right. A true constant will make it rotate the array to the left.

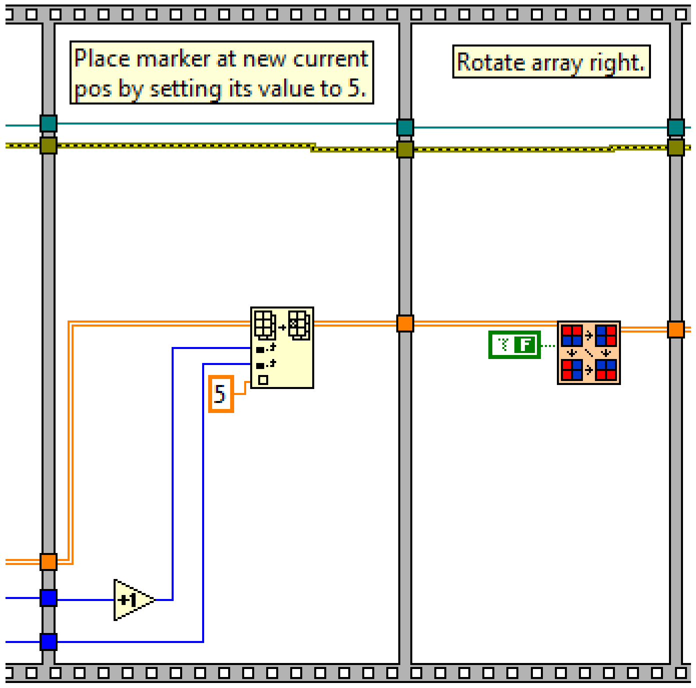


Fig. 6-5

**Step 10**: After the array has been rotated we now need to relocate where the robot is in the array. Thus, we must search each element in the array for the value of 5 or whatever number you have set as the marker. Normally two for loops are used to search through the elements of an array but since we need to keep the values of the row and col where the marker value was found, we need to exit the loops as soon as we find that value. Thus, two while loops are used where the stop condition is set to check whether the loop iteration has reached the last col or row or whether it has found the marker value. Once found, the value of the loop iteration of the outer loop will be the new row and the value of the loop iteration of the inner loop will be the new col.

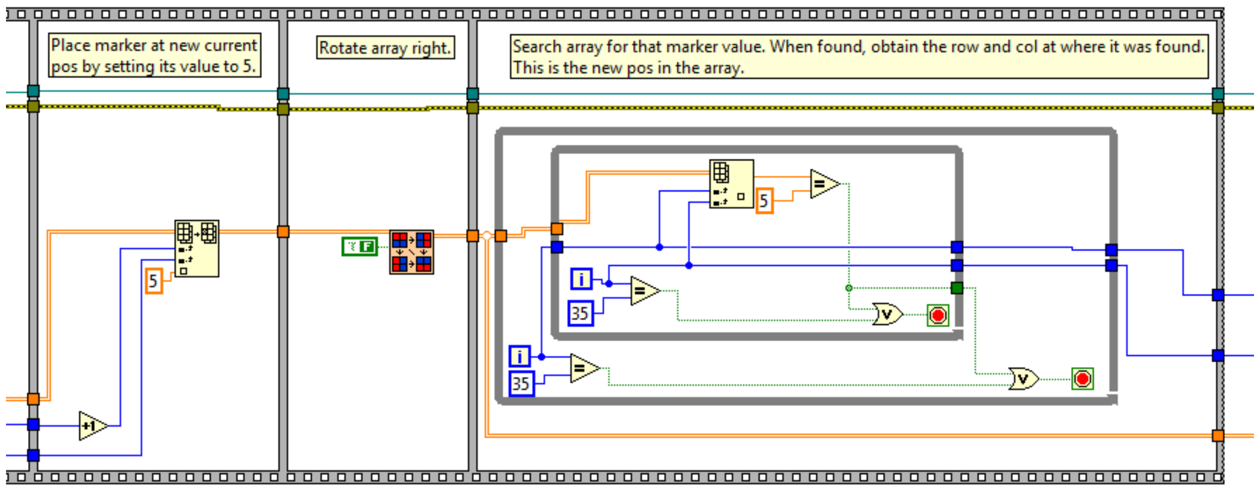

Place marker at new current pos by setting its value to 5.
Rotate array right.
Search array for that marker value. When found, obtain the row and col at where it was found. This is the new pos in the array.
5
+1
F
5
i
35
i
35

Fig. 6-6

**Step 11**: Once you have wired the outputs of the flat sequence to the end of the case statement, you now go through the same procedure with the case statement for checking the left position. Remember that a true Boolean constant wired to the “Rotate Array” subVI will make the subVI rotate the array to the left.

**Step 12**: Now we must configure the front diagonal right and front diagonal left case statements. This code will only make the robot move horizontally and vertically. Thus for these case statements, we must now check for whether the space in front of the robot is empty. If so, then the robot will move forward, turn right or left depending on the case statement, and then move forward again. Otherwise, the robot will turn to the right or left first, move forward, turn 90 degrees back to the right or left to make itself face forward, and then move forward again.

**Step 13**: Delete the flat sequence that is in the case statement that checks the front diagonal right position. First we will place an ”Index Array” subVI in the case statement and wire the array wire to its input along with the row and col values to make it return the value from the array element directly in front of the robot. The values for the row and col in front of the robot are same row value and col +1. We will then wire that output to a case statement. The case statement will have two cases and the default case can be either one of them. There will be a case where the position in front of the robot is 1 and another for when it’s 100.

**Step 14**: In the case where the position in front of the robot is 100, place a flat sequence in the array and do the necessary steps to make the robot turn to the right 90 degrees, move forward, turn back to the left 90 degrees, move forward.

**Step 15**: The case for when the position in front of the robot is 1 is a bit more involved. Do the necessary steps to make the robot move forward, turn 90 degrees to the right, and move forward again but do not make it turn back to the left 90 degrees.

**Step 16**: Now we do the same procedure we did for the end of the flat sequence in the case statement where we check the robot’s right position. Those steps are: change the value of the array element that the robot is moving towards to the marker value, rotate the array to the right, search the rotated array for the marker value, and once found pass the outputs of the sequence to the end of the case statement for checking the front position of the robot and pass those outputs to the end of the case statement checking the front diagonal right position.

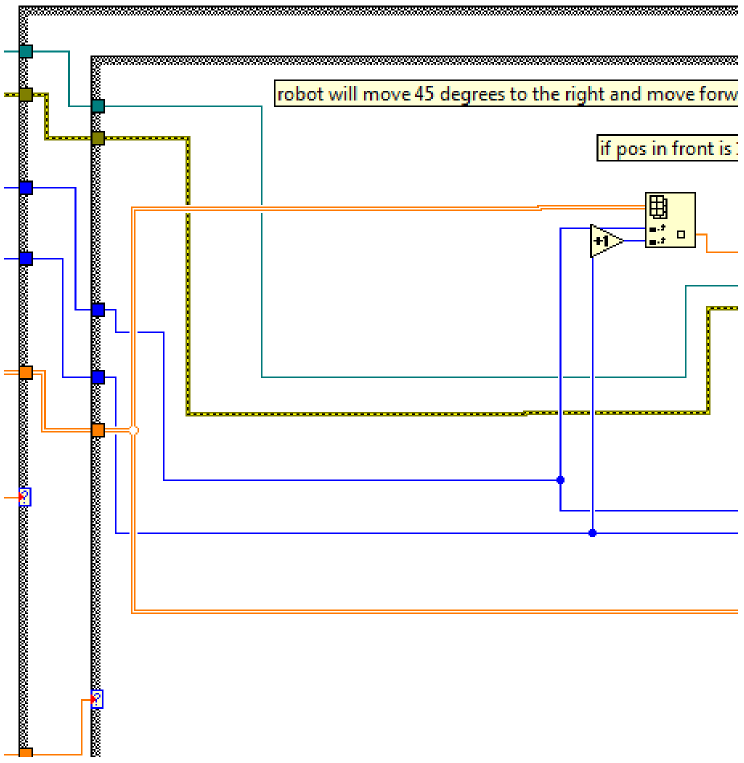


Fig. 6-7

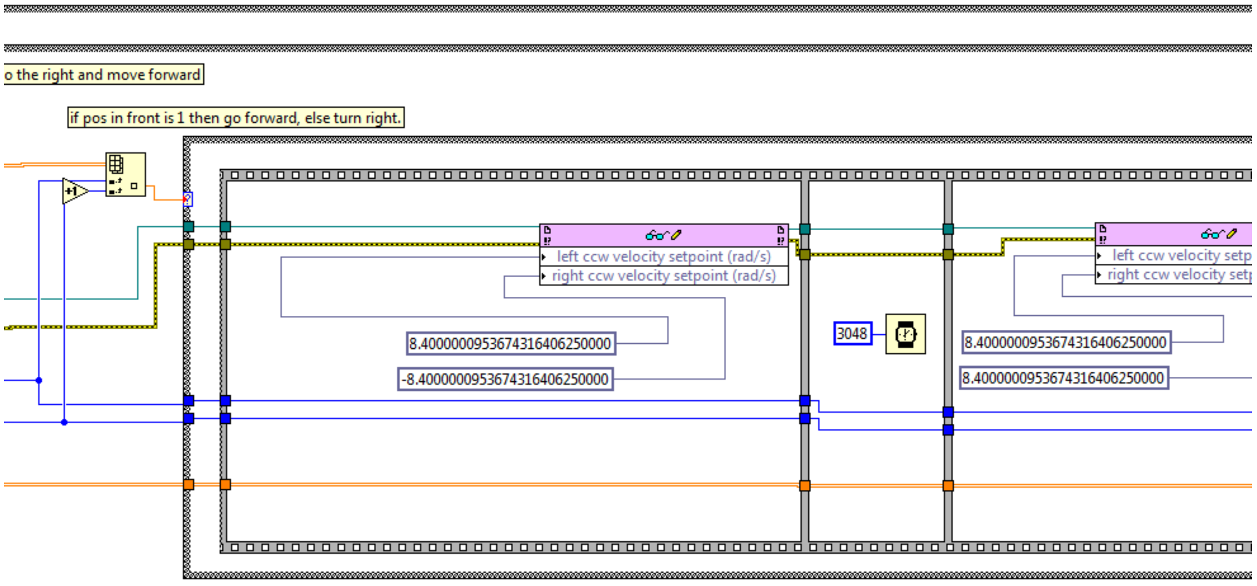


Fig. 6-8

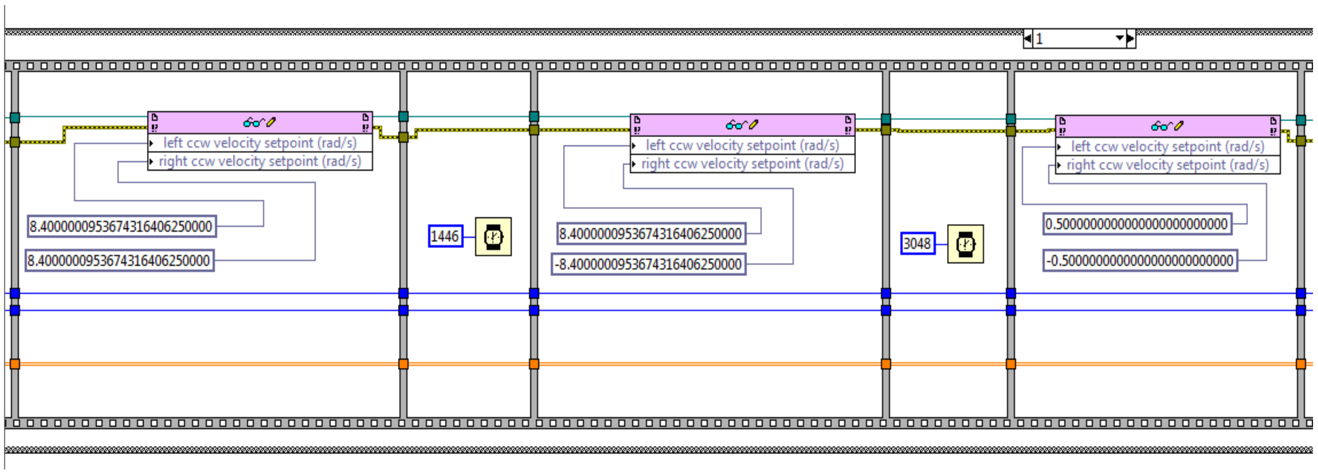

1
left ccw velocity setpoint (rad/s)
right ccw velocity setpoint (rad/s)
8.4000000953674316406250000
8.4000000953674316406250000
1446
left ccw velocity setpoint (rad/s)
right ccw velocity setpoint (rad/s)
8.4000000953674316406250000
-8.4000000953674316406250000
3048
left ccw velocity setpoint (rad/s)
right ccw velocity setpoint (rad/s)
0.5000000000000000000000000
-0.5000000000000000000000000

Fig. 6-9

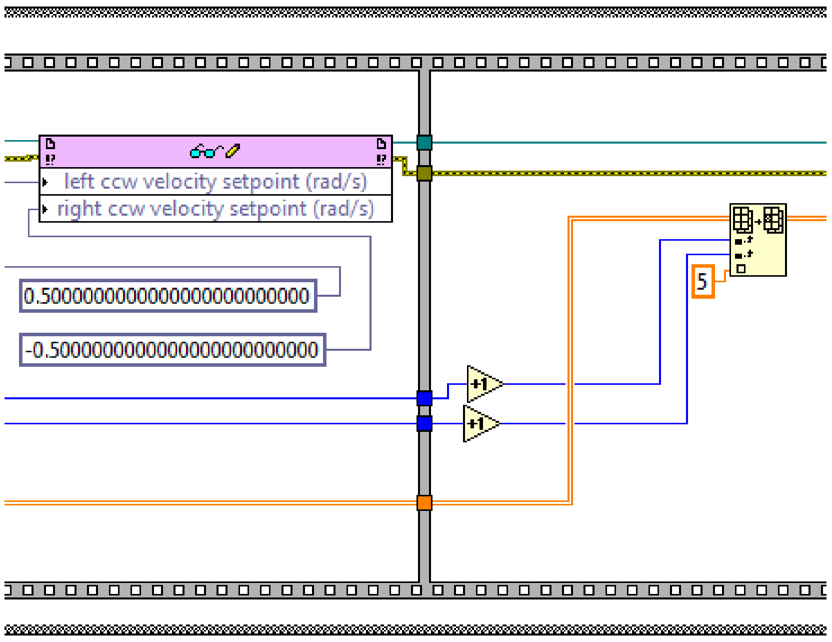

left ccw velocity setpoint (rad/s)
right ccw velocity setpoint (rad/s)
0.5000000000000000000000000
-0.5000000000000000000000000
+1
+1
5

Fig. 6-10

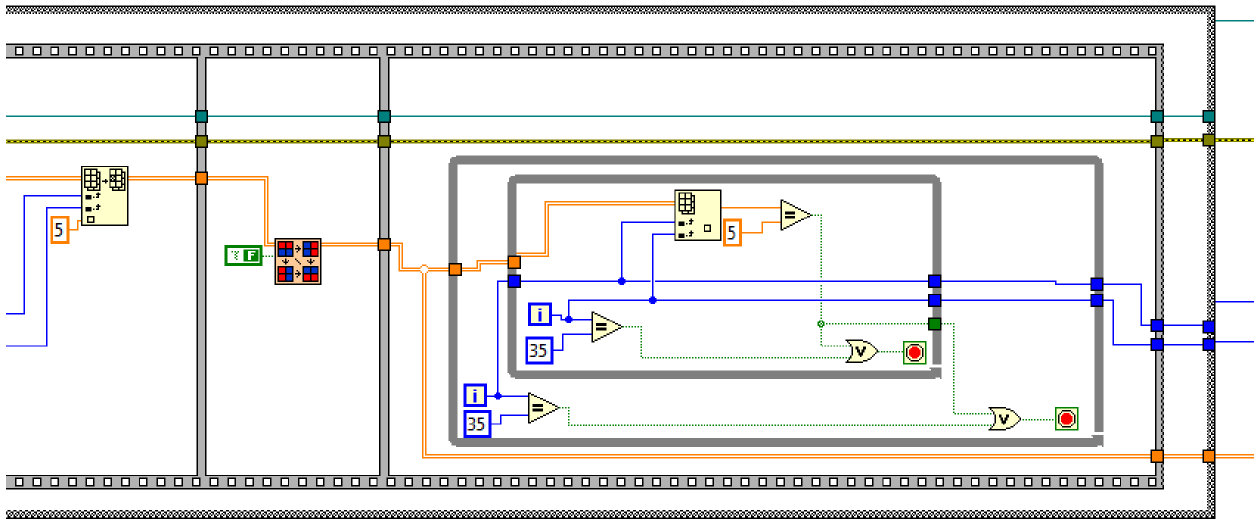


Fig. 6-11

**Step 17**: Again, this same procedure is used for the case statement that checks front diagonal left position. Remember that to rotate the array to the left you must wire a true Boolean constant to the "Rotate Array" subVI.

**Step 18**: Once this is done, save the file, build the "Obstacle Course" VI, and run as startup and you are done with this lab.