1 Introduction

The objective of this project was to develop a hardware-in-the-loop testbed on for the robot race car, enabling end-to-end testing of the robot without any modifications to the robot's software. This was accomplished by developing software for a test controller (the plant) on top of a real-time operating system that simulates the race track and emulates the robot's sensors and actuators. As the robot “moves” through the simulated environment, the sensor values produced by the test controller are updated and sent to the robot controller through the robot's normal sensor input pins. The robot then responds to the sensor values by modifying its actuator values, which are then read by the test controller using the robot's normal actuator output pins. The test controller then uses these actuator values to modify the robot's simulated position, velocity, and direction in the environment. The sensor values produced by the test controller are updated to reflect these modifications, and the cycle continues until the robot passes the finish line or hits a wall. There were three key components to this project: the environment simulation software, the sensor and actuator emulators, and the hardware integration between the test and robot controllers. The software can be found here in the following repository:

https://github.com/smasimore/HILRacecarTesting/blob/master/HILMain.c

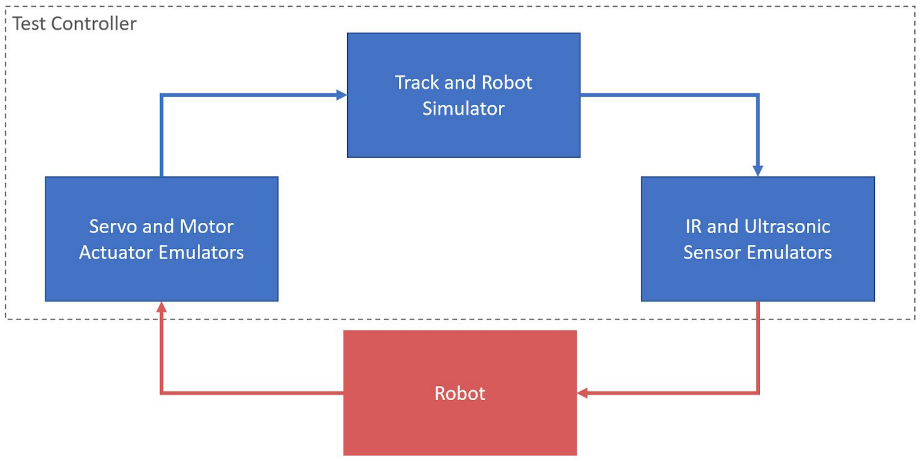

Test Controller
Track and Robot Simulator
Servo and Motor Actuator Emulators
IR and Ultrasonic Sensor Emulators
Robot

Figure 1. Hardware-in-the-loop structure for racing robot.

2 Track and Robot Simulation Software

2.2 Simulating the Environment

To simulate the environment, three components are defined: the track boundaries, the walls within the track, and the finish line. Figure 3 is one example of a configuration used to test the simulation.

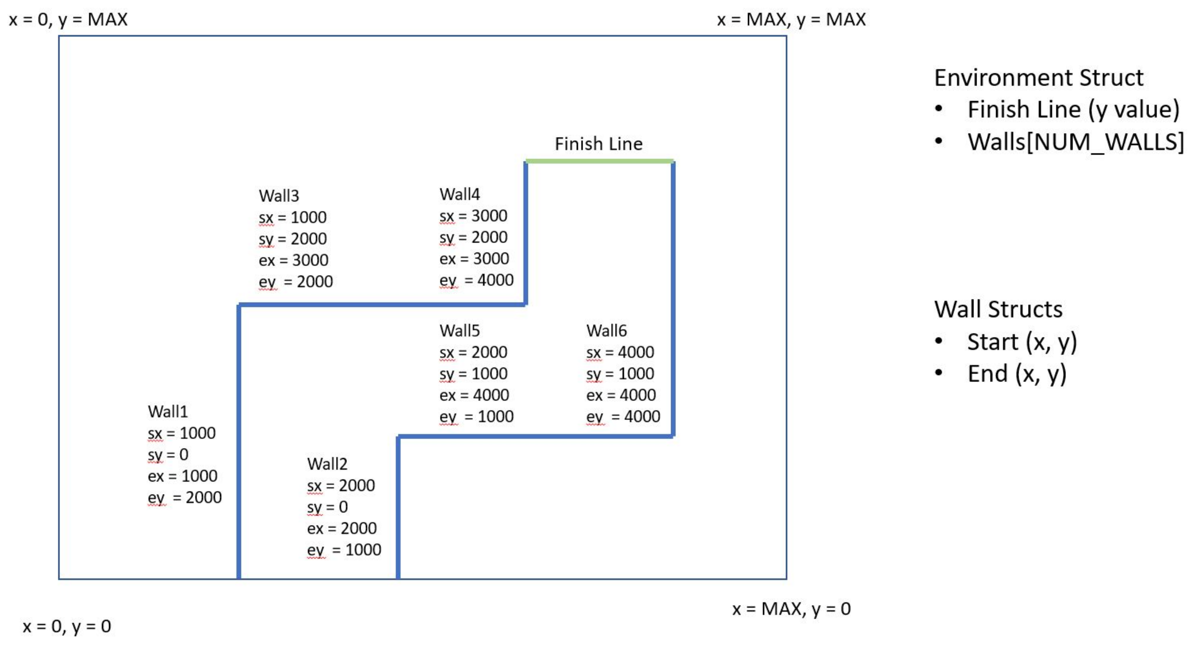
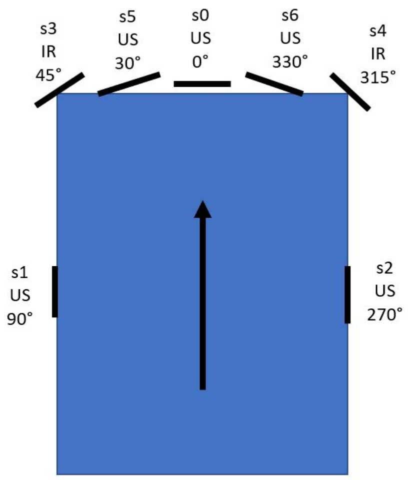


Figure 2. Example environment configuration.

2.4 Simulating the Robot

The robotis represented as a single point in the environment. The robot has a position (x, y in mm from 0, 0 to MAX, MAX), velocity (mm/s), and direction (in degrees from 0 to 360 where 0 is going in increasing order down the x-axis and 90 is going in increasing order up the y-axis). The robot has an array of Sensor structs that represent each sensor on the robot, including the sensor type (infrared or ultrasonic), sensor ID (used to determine what pins are used in emulation), position (degrees relative to robot), and current distance from nearest wall (mm).



Car Struct
- Position (x, y) relative to environment
- Velocity (mm/s)
- Direction (deg), relative to bottom boundary of environment
- Sensors[NUM_SENSORS]
- Car represented as single x, y position in Environment

Sensor Structs
- ID
- Type (IR vs. US)
- Direction in degrees, relative to direction of car

Figure 3. Example Robot/Car and Sensor configuration.

2.3 Managing the Robot's Movement

A state machine is used to manage and keep track of the robot's movement through the simulation and update the robot's sensor values.

Each state contains the following variables:

- Time
- Car's x, y position
- Car's velocity and direction
- Each sensor's distance from nearest wall

The input for each state transition is:

- Car's servo value (steering)
- Car's motor value (velocity)

To transition between states a hardware timer interrupts according to the defined SIM_FREQ constant. Each transition consists of the following steps:

1. Read actuator values
2. Log row to SimLogger for dumping to terminal at end of test
3. Update robot’s position in environment based on actuator values and time increment. Calculate the distance traveled and determine the change in x and y position using scaled trig lookup tables and fixed-point math.

```
hyp = vel * timePassedMs / 1000;
deltaX = CosLookup[dir] * hyp / TRIG_SCALE;
deltaY = SinLookup[dir] * hyp / TRIG_SCALE;
```

4. Check if robot has hit a wall. If so, end test with failure. This is done by determining if the line segment formed by the robot’s previous position and new position intersect with any walls.
5. Check if robot has crossed the finish line. If so, end test with success
6. Update the robot’s sensor values by determining the sensor’s line-of-sight and finding the minimum distance from an intersecting wall. Similar to checking if the robot hit a wall, this calculation uses scaled trig lookup tables and fixed-point math.

1. Calculate sensor’s absolute angle relative to environment
2. Uses fixed-point math and sin/cos lookup table to calculate start (x, y) and end (x, y) of sensor’s line-of-sight
3. Iterates through environments walls, for each wall:
    1. Uses fixed-point math to determine if and where the sensor’s line-of-sight and the wall segment intersect
    2. Calculates distance from wall
    3. Updates minimum distance

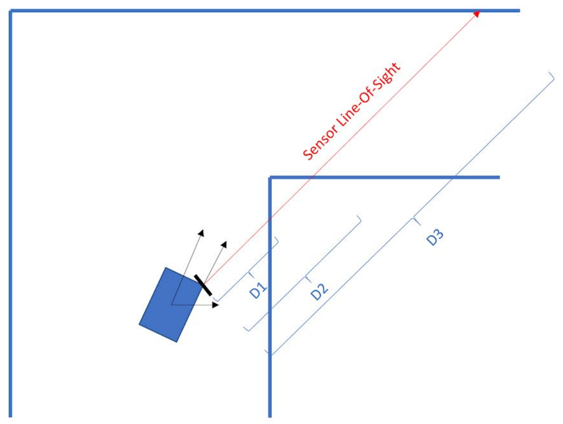


Figure 4. Calculating sensor distance from nearest wall.

7. Update sensor emulators with new values

2.4 Software Organization

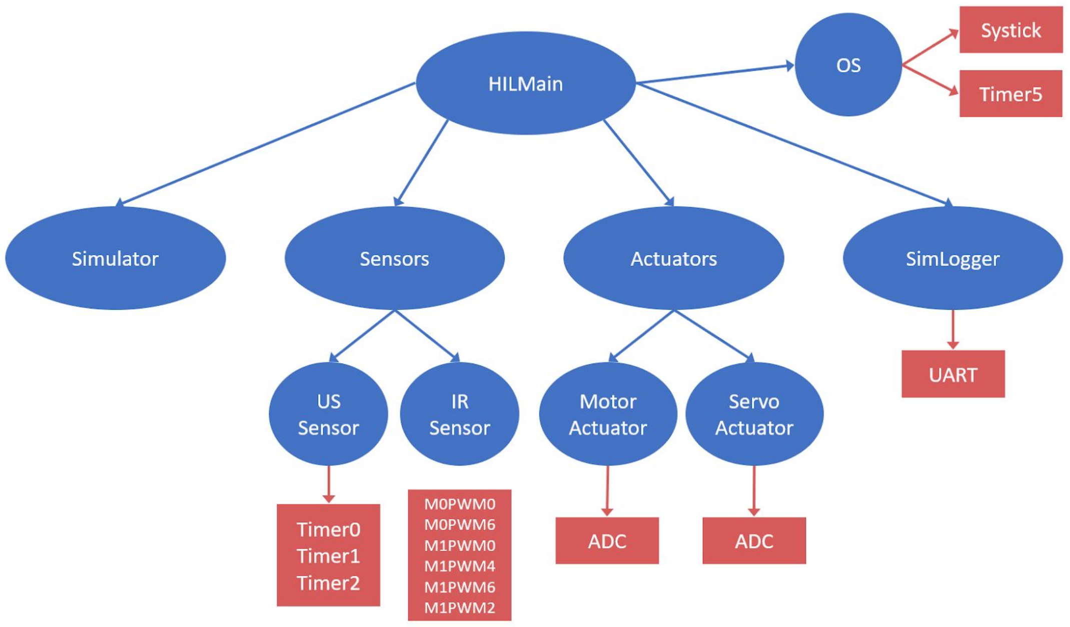


Figure 5. Call graph for test controller.

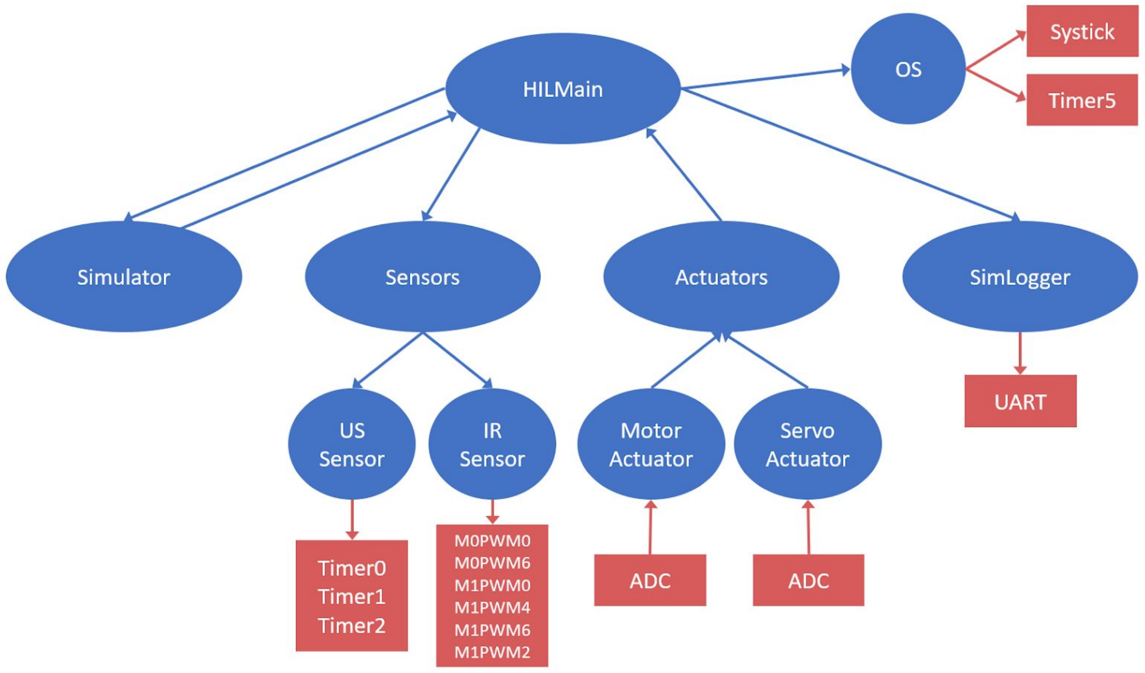


Figure 6. Data flow graph for test controller.

2.4 Simulation Verification

To verify the simulator worked as expected, an integration test was run on 3 different configurations with actuator inputs mocked. Mocking the actuator inputs allows for controlling the path the robot took. The test results were then compared to expected values.

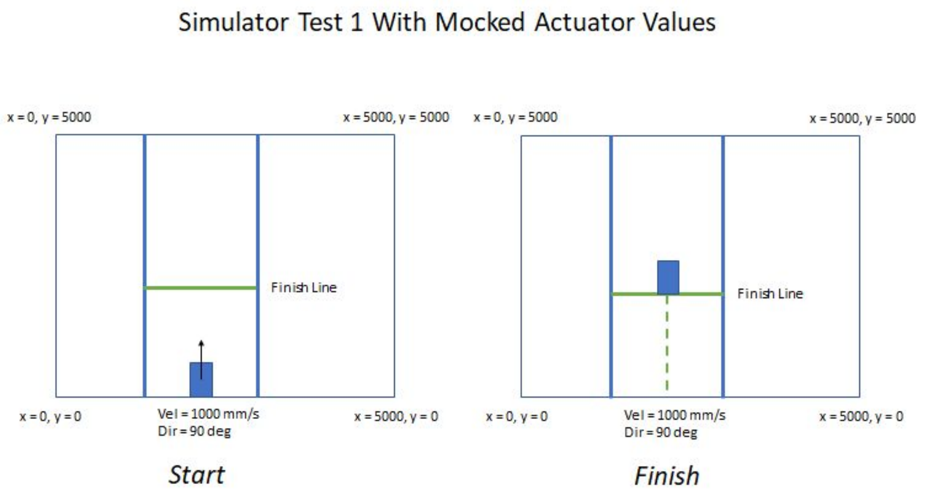
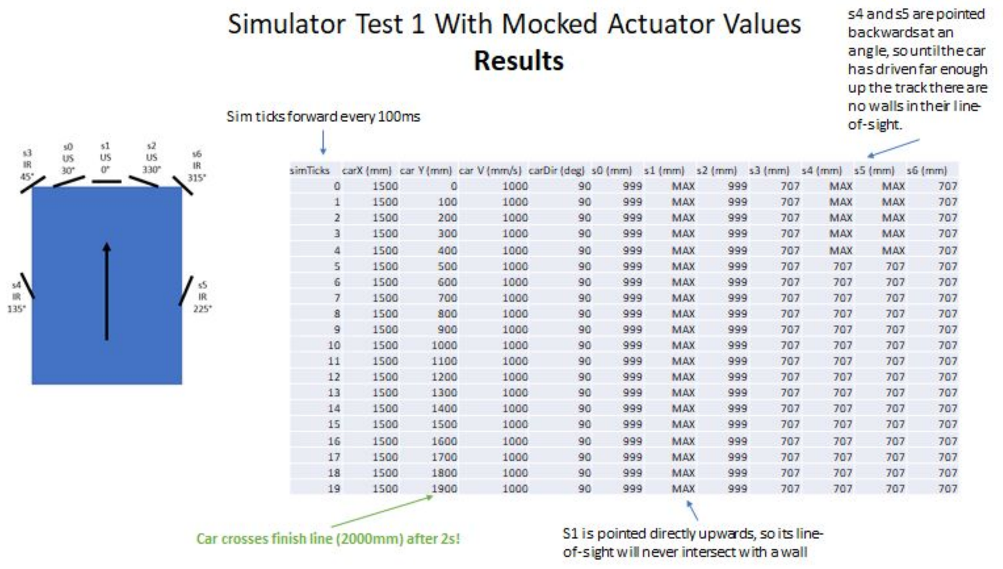


Figure 7. Simulator Integration Test 1 configuration


Simulator Test 1 With Mocked Actuator Values
Results
s3
IR
45°
s0
US
30°
s1
US
0°
s2
US
330°
s6
IR
315°
s4
IR
135°
s5
IR
225°

s4 and s5 are pointed backwards at an angle, so until the car has driven far enough up the track there are no walls in their line-of-sight.

Sim ticks forward every 100ms

simTicks	carX (mm)	car Y (mm)	car V (mm/s)	carDir (deg)	s0 (mm)	s1 (mm)	s2 (mm)	s3 (mm)	s4 (mm)	s5 (mm)	s6 (mm)
0	1500	0	1000	90	999	MAX	999	707	MAX	MAX	707
1	1500	100	1000	90	999	MAX	999	707	MAX	MAX	707
2	1500	200	1000	90	999	MAX	999	707	MAX	MAX	707
3	1500	300	1000	90	999	MAX	999	707	MAX	MAX	707
4	1500	400	1000	90	999	MAX	999	707	MAX	MAX	707
5	1500	500	1000	90	999	MAX	999	707	707	707	707
6	1500	600	1000	90	999	MAX	999	707	707	707	707
7	1500	700	1000	90	999	MAX	999	707	707	707	707
8	1500	800	1000	90	999	MAX	999	707	707	707	707
9	1500	900	1000	90	999	MAX	999	707	707	707	707
10	1500	1000	1000	90	999	MAX	999	707	707	707	707
11	1500	1100	1000	90	999	MAX	999	707	707	707	707
12	1500	1200	1000	90	999	MAX	999	707	707	707	707
13	1500	1300	1000	90	999	MAX	999	707	707	707	707
14	1500	1400	1000	90	999	MAX	999	707	707	707	707
15	1500	1500	1000	90	999	MAX	999	707	707	707	707
16	1500	1600	1000	90	999	MAX	999	707	707	707	707
17	1500	1700	1000	90	999	MAX	999	707	707	707	707
18	1500	1800	1000	90	999	MAX	999	707	707	707	707
19	1500	1900	1000	90	999	MAX	999	707	707	707	707

Car crosses finish line (2000mm) after 2s!

S1 is pointed directly upwards, so its line-of-sight will never intersect with a wall

Figure 8. Simulator Integration Test 1 results (matches expected).

Simulator Test 2 With Mocked Actuator Values

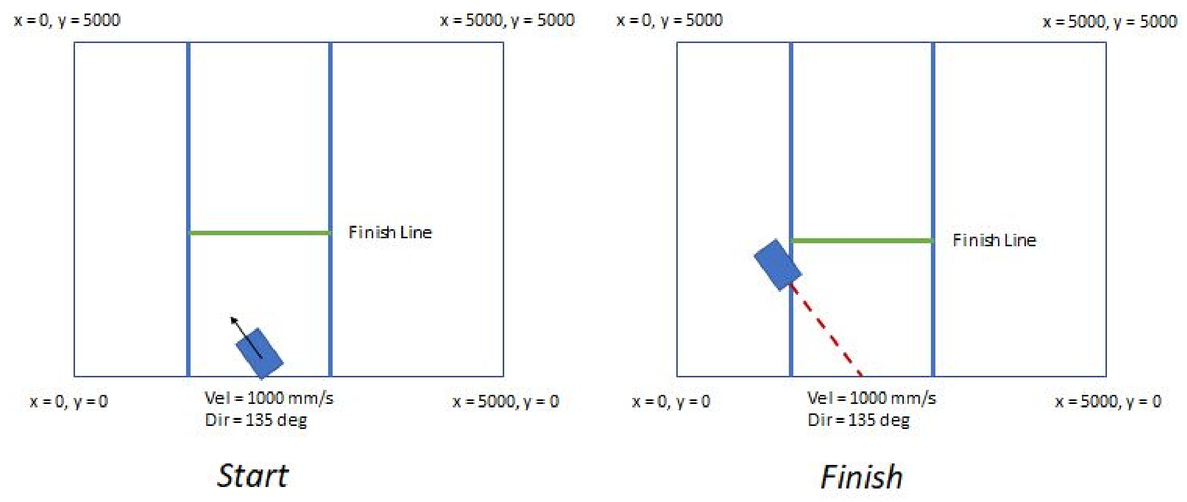
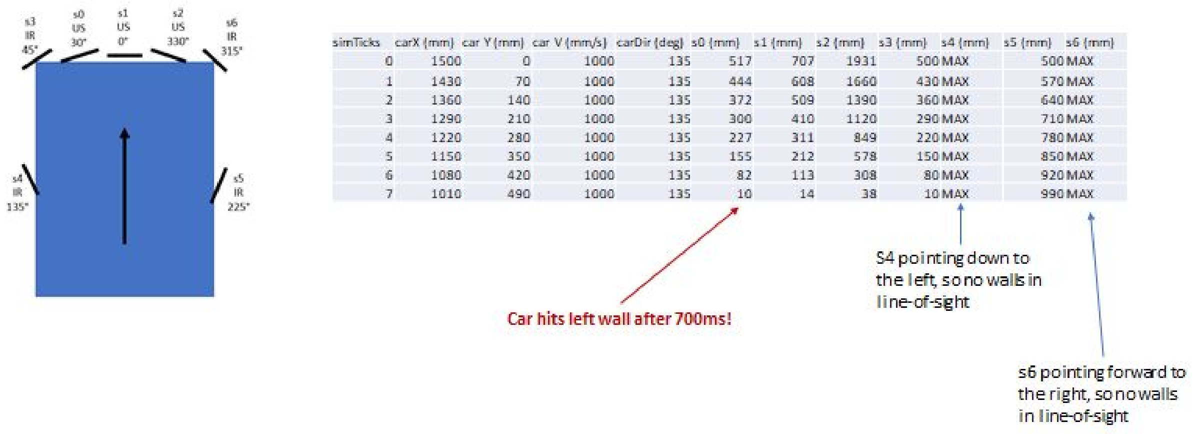


Figure 9. Simulator Integration Test 2 configuration

Simulator Test 2 With Mocked Actuator Values
Results


s3 IR 45°
s0 US 30°
s1 US 0°
s2 US 330°
s6 IR 315°
s4 IR 135°
s5 IR 225°
Car hits left wall after 700ms!
S4 pointing down to the left, so no walls in line-of-sight
s6 pointing forward to the right, so no walls in line-of-sight

simTicks	carX (mm)	car Y (mm)	car V (mm/s)	carDir (deg)	s0 (mm)	s1 (mm)	s2 (mm)	s3 (mm)	s4 (mm)	s5 (mm)	s6 (mm)
0	1500	0	1000	135	517	707	1931	500	MAX	500	MAX
1	1430	70	1000	135	444	608	1660	430	MAX	570	MAX
2	1360	140	1000	135	372	509	1390	360	MAX	640	MAX
3	1290	210	1000	135	300	410	1120	290	MAX	710	MAX
4	1220	280	1000	135	227	311	849	220	MAX	780	MAX
5	1150	350	1000	135	155	212	578	150	MAX	850	MAX
6	1080	420	1000	135	82	113	308	80	MAX	920	MAX
7	1010	490	1000	135	10	14	38	10	MAX	990	MAX

Figure 10. Simulator Integration Test 2 results (matches expected).

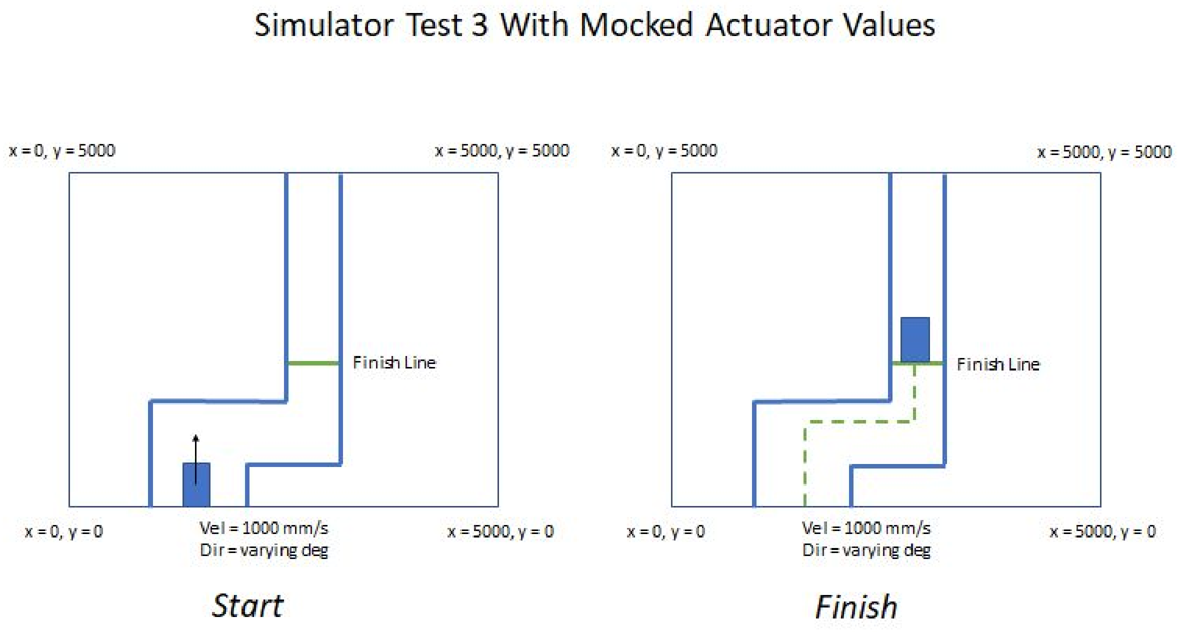
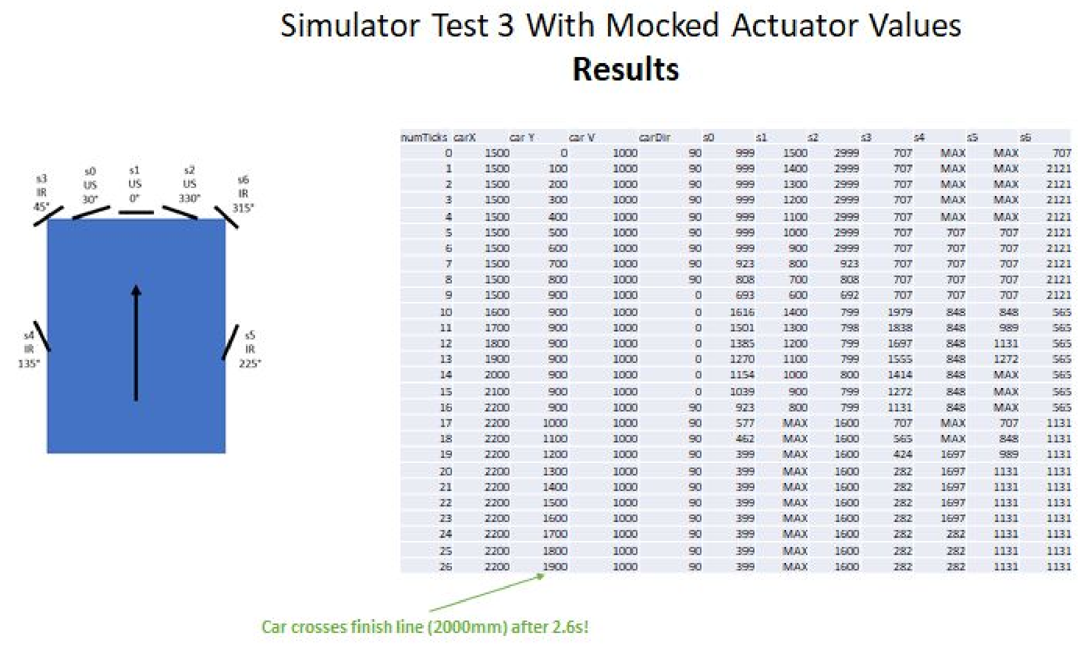


Figure 11. Simulator Integration Test 3 configuration


Simulator Test 3 With Mocked Actuator Values
Results
s3 IR 45°
s0 US 30°
s1 US 0°
s2 US 330°
s6 IR 315°
s4 IR 135°
s5 IR 225°
Car crosses finish line (2000mm) after 2.6s!

numTicks	carX	car Y	car V	carDir	s0	s1	s2	s3	s4	s5	s6
0	1500	0	1000	90	999	1500	2999	707	MAX	MAX	707
1	1500	100	1000	90	999	1400	2999	707	MAX	MAX	2121
2	1500	200	1000	90	999	1300	2999	707	MAX	MAX	2121
3	1500	300	1000	90	999	1200	2999	707	MAX	MAX	2121
4	1500	400	1000	90	999	1100	2999	707	MAX	MAX	2121
5	1500	500	1000	90	999	1000	2999	707	707	707	2121
6	1500	600	1000	90	999	900	2999	707	707	707	2121
7	1500	700	1000	90	923	800	923	707	707	707	2121
8	1500	800	1000	90	808	700	808	707	707	707	2121
9	1500	900	1000	0	693	600	692	707	707	707	2121
10	1600	900	1000	0	1616	1400	799	1979	848	848	565
11	1700	900	1000	0	1501	1300	798	1838	848	989	565
12	1800	900	1000	0	1385	1200	799	1697	848	1131	565
13	1900	900	1000	0	1270	1100	799	1555	848	1272	565
14	2000	900	1000	0	1154	1000	800	1414	848	MAX	565
15	2100	900	1000	0	1039	900	799	1272	848	MAX	565
16	2200	900	1000	90	923	800	799	1131	848	MAX	565
17	2200	1000	1000	90	577	MAX	1600	707	MAX	707	1131
18	2200	1100	1000	90	462	MAX	1600	565	MAX	848	1131
19	2200	1200	1000	90	399	MAX	1600	424	1697	989	1131
20	2200	1300	1000	90	399	MAX	1600	282	1697	1131	1131
21	2200	1400	1000	90	399	MAX	1600	282	1697	1131	1131
22	2200	1500	1000	90	399	MAX	1600	282	1697	1131	1131
23	2200	1600	1000	90	399	MAX	1600	282	1697	1131	1131
24	2200	1700	1000	90	399	MAX	1600	282	282	1131	1131
25	2200	1800	1000	90	399	MAX	1600	282	282	1131	1131
26	2200	1900	1000	90	399	MAX	1600	282	282	1131	1131

Figure 12. Simulator Integration Test 3 results (matches expected).

3 Sensor Emulators

3.1 Ping Sensor Emulator

The test controller emulates up to 3 ultrasonic Ping sensors. A Ping sensor works by sending an ultrasonic pulse and timing the amount of time it takes for reflections to be received. To emulate the sensors, the test controller must listen for this pulse, wait a certain amount of time, and send a response. To accomplish this each Ping emulator is initialized with a GPIO pin and timer. The GPIO pins are initially set up as inputs with an interrupt on a rising edge. The timers are set up as one-shot timers and initially unarmed. When the GPIO pin receives a pulse from the robot, the edge-triggered interrupt is handled. The handler:

1. Acknowledges the interrupt
2. Disarms the GPIO interrupt for the specific pin that triggered the interrupt
3. Sets the respective timer period to be PING_HOLDOFF (given in Ping datasheet)
4. Arms the timer

Once the holdoff time elapses, the relevant timer interrupts. The handler:

1. Sets the GPIO pin as output
2. Sends a high signal over the pin
3. Sets timer period to be time required to emulate specified distance

The timer will then go off a second time. This time the handler:

1. Sets the GPIO pin as input
2. Sets the pin low
3. Arms the GPIO edge-triggered interrupt
4. Disables the timer

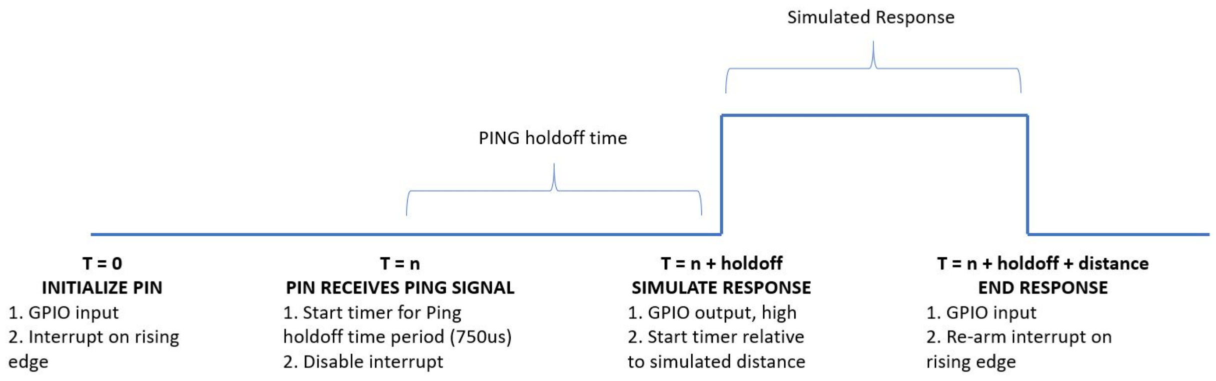


Figure 13. Ping ultrasonic emulator steps.

When emulating the sensor, the simulator provides the sensor distance to emulate. To get the timer period from this distance, the following formula is used:

```
Period = CLOCK_FREQ / MACH_MM_PER_SECOND * val * 2; // x2 to account for back and forth
```

3.2 IR Sensor Emulator

The test controller emulates up to 5 IR sensors. The IR sensor works by providing a voltage to the robot that corresponds to a distance. To emulate the IR sensor, each IR sensor is initialized with a PWM channel to produce a voltage relative to the desired distance to emulate. Because the robot is expecting an analog signal rather than the digital signal produced by the PWM channel, an RC circuit is used to smooth out the signal.

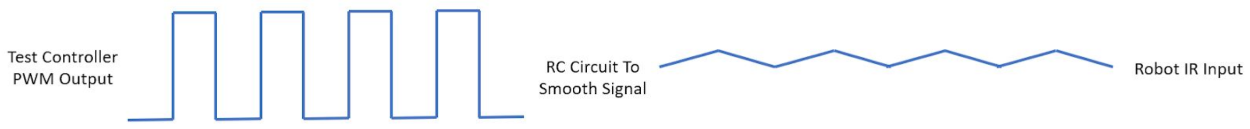


Figure 14. IR sensor emulation signal smoothing.

To calculate voltage to produce from the desired distance to emulate, a lookup table was created mapping mm to duty cycle for PWM. This lookup table is offset by the minimum possible distance the IR sensors can sense.

3.3 Sensor Verification

After calibrating the Ping/US and IR sensors an integration test was done with the robot to verify correctness.

Test Distance (mm)	**IR0 Actual (mm)**	**IR1 Actual (mm)**	**IR2 Actual (mm)**	**IR3 Actual (mm)**	**US0 Actual (mm)**	**US1 Actual (mm)**	**US2 Actual (mm)**
100	113	111	112	113	99	99	99
200	210	209	212	213	199	199	199
300	313	306	301	301	299	299	299
400	408	402	399	399	398	398	398
500	500	497	493	504	498	498	498
600	627	602	590	588	598	598	598
700	0	0	0	0	697	697	697
1000	0	0	0	0	997	996	997
1500	0	0	0	0	1495	1495	1495

Sensor test observations:

- IR sensor emulation results in a higher level of variability than the US sensor emulation
- The IR sensor variability is higher at lower distance emulations (~10% variation from target vs. ~2% variation at higher distances)
- The robot cannot read values from the IR sensor above 600mm
- The robot cannot read values from the US sensor above 1500mm

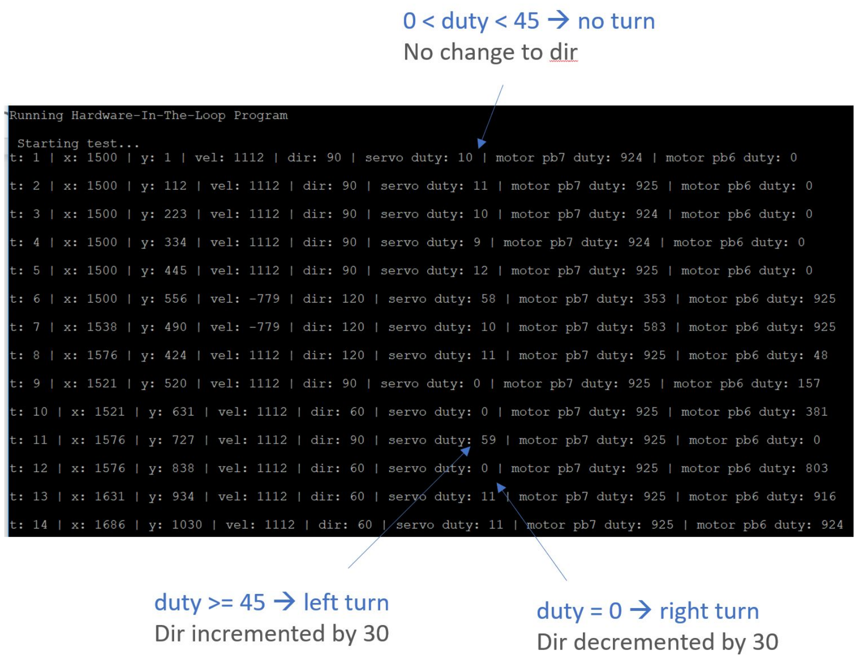
4 Actuator Emulators

4.1 Servo Actuator Emulator

A servo is used by the robot to control steering. The robot produces a PWM signal to control the servo. To emulate the servo the robot's PWM signal is passed through an RC circuit to smooth the signal and read by a GPIO pin on the test controller. The car state was then categorized as turning right, turning left, or going straight. These 3 bins were chosen due to low resolution of the servo signal (this can be improved by using a DAC → ADC circuit instead of the PWM → RC → ADC circuit). Because the servo is controlled using low voltages, a higher resistor and capacitor is used (vs. the IR sensor) to smooth the PWM signal.

4.2 Verifying Servo Emulator

To verify the behavior of the servo emulator the MotorBoard was set to maintain right, mid, and left servo directions and the value read at the test controller was compared to expected. The voltage to direction conversion was calibrated with the actual servo actuator running parallel (this changes the voltage reading base). Additional system-level tests were run with the entire system running to verify servo emulation.



```
Running Hardware-In-The-Loop Program

 Starting test...
t: 1 | x: 1500 | y: 1 | vel: 1112 | dir: 90 | servo duty: 10 | motor pb7 duty: 924 | motor pb6 duty: 0

t: 2 | x: 1500 | y: 112 | vel: 1112 | dir: 90 | servo duty: 11 | motor pb7 duty: 925 | motor pb6 duty: 0

t: 3 | x: 1500 | y: 223 | vel: 1112 | dir: 90 | servo duty: 10 | motor pb7 duty: 924 | motor pb6 duty: 0

t: 4 | x: 1500 | y: 334 | vel: 1112 | dir: 90 | servo duty: 9 | motor pb7 duty: 924 | motor pb6 duty: 0

t: 5 | x: 1500 | y: 445 | vel: 1112 | dir: 90 | servo duty: 12 | motor pb7 duty: 925 | motor pb6 duty: 0

t: 6 | x: 1500 | y: 556 | vel: -779 | dir: 120 | servo duty: 58 | motor pb7 duty: 353 | motor pb6 duty: 925

t: 7 | x: 1538 | y: 490 | vel: -779 | dir: 120 | servo duty: 10 | motor pb7 duty: 583 | motor pb6 duty: 925

t: 8 | x: 1576 | y: 424 | vel: 1112 | dir: 120 | servo duty: 11 | motor pb7 duty: 925 | motor pb6 duty: 48

t: 9 | x: 1521 | y: 520 | vel: 1112 | dir: 90 | servo duty: 0 | motor pb7 duty: 925 | motor pb6 duty: 157

t: 10 | x: 1521 | y: 631 | vel: 1112 | dir: 60 | servo duty: 0 | motor pb7 duty: 925 | motor pb6 duty: 381

t: 11 | x: 1576 | y: 727 | vel: 1112 | dir: 90 | servo duty: 59 | motor pb7 duty: 925 | motor pb6 duty: 0

t: 12 | x: 1576 | y: 838 | vel: 1112 | dir: 60 | servo duty: 0 | motor pb7 duty: 925 | motor pb6 duty: 803

t: 13 | x: 1631 | y: 934 | vel: 1112 | dir: 60 | servo duty: 11 | motor pb7 duty: 925 | motor pb6 duty: 916

t: 14 | x: 1686 | y: 1030 | vel: 1112 | dir: 60 | servo duty: 11 | motor pb7 duty: 925 | motor pb6 duty: 924
```

Figure 15. Servo emulator test.

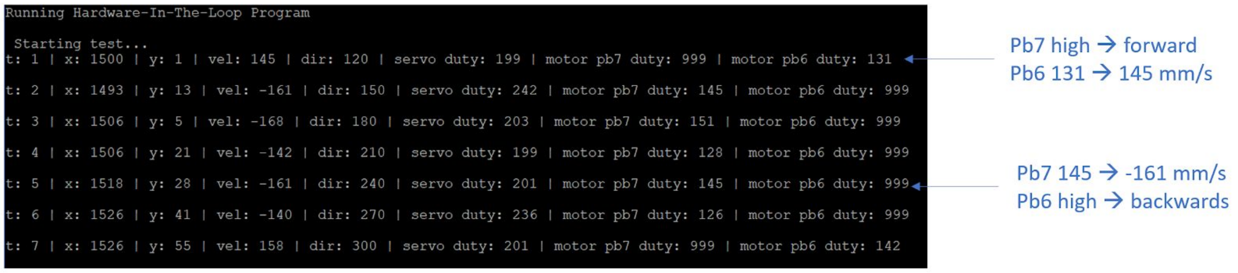
4.3 Motor Actuator Emulator

There are 2 motors used to give the robot velocity, a left and right. To reduce the complexity of this project only one motor is monitored and differential motor speeds are ignored. To emulate the motor 2 pins are read from the robot into ADC pins on the test controller. When moving forward, one pin is high while the other is a voltage controlled by a PWM on the robot. When moving backwards, the pin roles flip. To calibrate the voltage to velocity conversion the robot was set to different PWM duty cycles and the velocity was measured on a flat, wood floor.

4.4 Verifying Motor Emulator

To verify the motor emulator the PWM output of the motor pins were measured throughout a test run and the resulting test velocity was compared to expected results. The actual motor and servo were *not* run in parallel for this test because the PWM channels read 0 in that case.

Motor Emulator Test



```
Running Hardware-In-The-Loop Program

 Starting test...
t: 1 | x: 1500 | y: 1 | vel: 145 | dir: 120 | servo duty: 199 | motor pb7 duty: 999 | motor pb6 duty: 131

t: 2 | x: 1493 | y: 13 | vel: -161 | dir: 150 | servo duty: 242 | motor pb7 duty: 145 | motor pb6 duty: 999

t: 3 | x: 1506 | y: 5 | vel: -168 | dir: 180 | servo duty: 203 | motor pb7 duty: 151 | motor pb6 duty: 999

t: 4 | x: 1506 | y: 21 | vel: -142 | dir: 210 | servo duty: 199 | motor pb7 duty: 128 | motor pb6 duty: 999

t: 5 | x: 1518 | y: 28 | vel: -161 | dir: 240 | servo duty: 201 | motor pb7 duty: 145 | motor pb6 duty: 999

t: 6 | x: 1526 | y: 41 | vel: -140 | dir: 270 | servo duty: 236 | motor pb7 duty: 126 | motor pb6 duty: 999

t: 7 | x: 1526 | y: 55 | vel: 158 | dir: 300 | servo duty: 201 | motor pb7 duty: 999 | motor pb6 duty: 142
```

Figure 16. Motor emulator test.

5 Hardware Integration

3.1 Pins Used on Test Controller

Component	**Pins Used**
Ping/US Sensor Emulator	PA3, PA4, PA5
IR Sensor Emulator	PB6, PC4, PF0, PF2, PE4
Servo Actuator	PE3
Motor Actuator	PE2, PE0

3.2 PWM → ADC Circuits

RC low pass filters were used to smooth the PWM signal to a cleaner analog signal for the IR sensor emulator and the servo and motor actuator emulators. The RC circuit varied based on the period of the PWM.

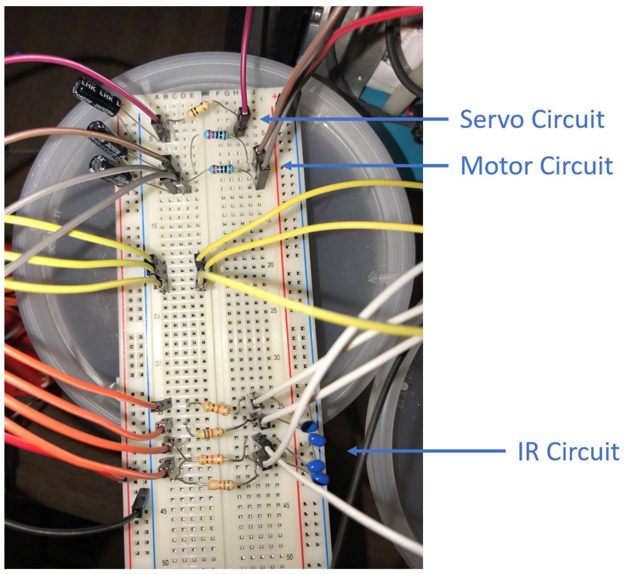


Figure 17. PWM → ADC Circuits.

6 Running the HIL Test

6.1 Final Setup

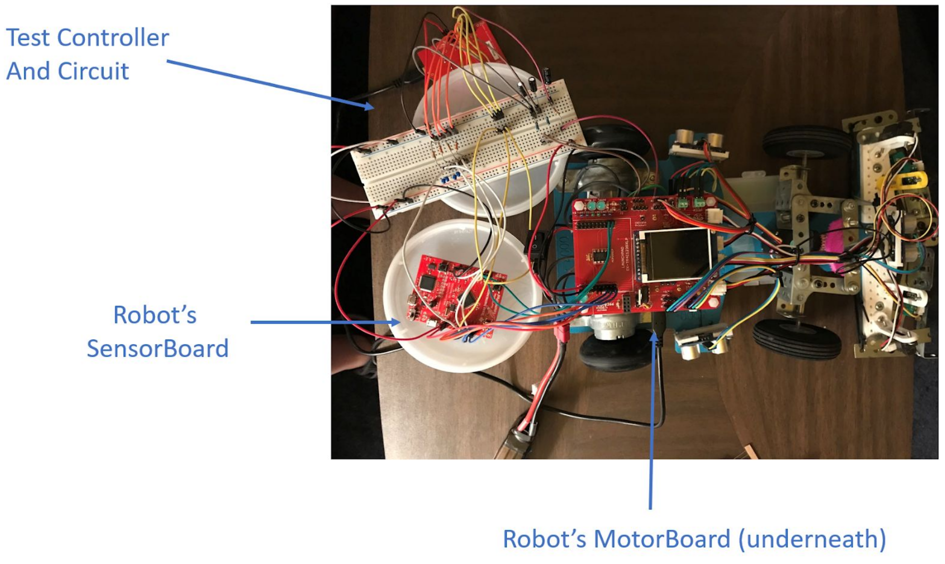


Figure 18. Final HIL setup with test controller and robot.

6.2 HIL Tests

During each test the test controller outputs live data for debugging. After the test is complete, the test log containing the robot's position and velocity and sensor readings at each simulation tick is printed out in csv format. Using Excel, the simulation is visualized. In the following examples the blue represents the track/walls, green is the finish line, and orange is the path the robot took.

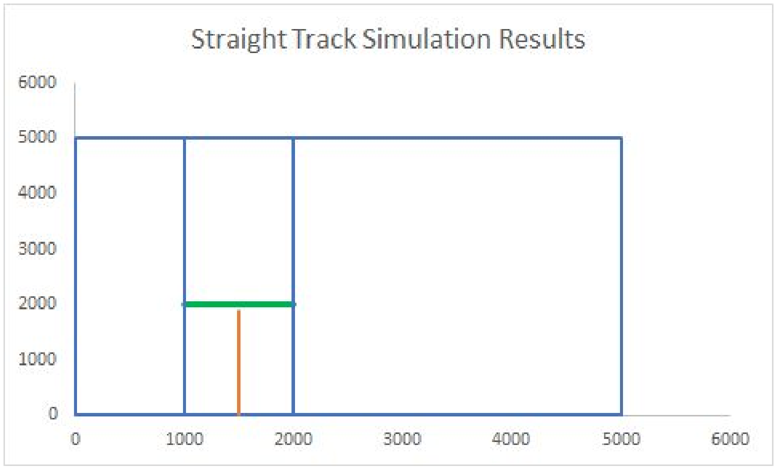


Figure 19. Straight track. Robot successfully completes track.

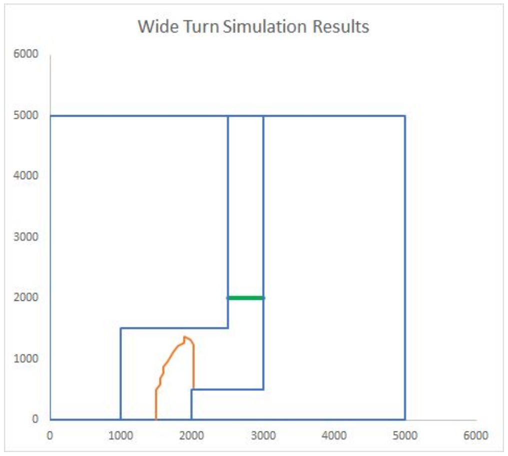


Figure 20. Wide turn track. Robot gets turned around and fails to completes track.

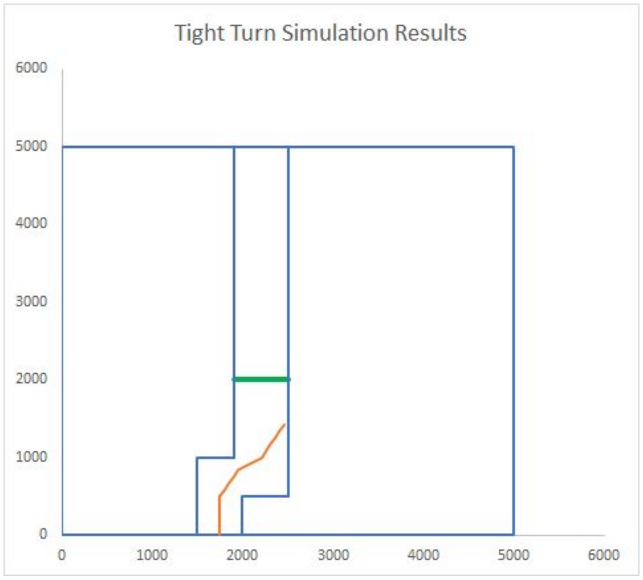


Figure 20. Tight turn track. Robot hits wall and fails to complete track.

7 Opportunities for Continued Work

The two main areas to improve this hardware-in-the-loop simulation is improving the hardware circuits so that the signals have higher resolution. Ideally the digital to analog conversion would be done by a DAC instead of using a low-pass filter to smooth the PWM signal. The second area is supporting more complexity in the simulator and emulators. This includes representing the car as a two-dimensional object instead of a point and supporting differential motor speeds and the impact that would have on direction and velocity.