Setup Manual

Project 25 – Responsive and Interactive Urban Track for Autonomous Vehicles

Team Members:

|  |  |  |
| --- | --- | --- |
| **Date** | **Revision** | **Release Notes** |
| 1/14/2024 | 01 | Initial Release |
| 4/27/24 | 02 | Final Release |
|  |  |  |
|  |  |  |

Table of Contents

[Product Overview Error! Bookmark not defined.](#_Toc164158750)

[Product Features Error! Bookmark not defined.](#_Toc164158751)

[Physical Track Error! Bookmark not defined.](#_Toc164158752)

[Given Equipment **Error! Bookmark not defined.**](#_Toc164158753)

[1. Product Overview 3](#_Toc164158754)

[2. Product Features 3](#_Toc164158755)

[3. Physical Track 4](#_Toc164158756)

[3.1 Given Equipment 4](#_Toc164158757)

[3.2 Track Layout Configurations 4](#_Toc164158758)

[3.2.1 Large Four Way 4](#_Toc164158759)

[3.2.2 Small Four Way 5](#_Toc164158760)

[3.2.3 Medium Roundabout 6](#_Toc164158761)

[Small Roundabout 7](#_Toc164158762)

[3.2.4 Full Oval 7](#_Toc164158763)

[3.2.5 Medium Oval 9](#_Toc164158764)

[3.2.6 Small Oval 10](#_Toc164158765)

[3.3 Barrier Configuration 10](#_Toc164158766)

[3.4 Interactive Element Configuration 11](#_Toc164158767)

[Stop Sign 11](#_Toc164158768)

[Yield Sign 11](#_Toc164158769)

[Stop Lights 12](#_Toc164158770)

[Railroad Crossing 12](#_Toc164158771)

[4. Software Implementation 13](#_Toc164158772)

[4.1 Software Products 13](#_Toc164158773)

[5. Example Setup 28](#_Toc164158774)

[5.1 Example Physical Track Layout 28](#_Toc164158775)

[5.2 Example Hardware Setup 29](#_Toc164158776)

[5.3 Example Software Setup 29](#_Toc164158777)

[6. Future Work 31](#_Toc164158778)

[6.1 More Track Layouts 31](#_Toc164158779)

# Product Overview

This product is an interactive urban track with physical and virtual world components created for small-scale autonomous cars. This documentation provides information on the given equipment and hardware and software setup.

# Product Features

This product includes multiple track layouts to test small-scale autonomous vehicles. The road features that are possible to encounter include a roundabout, a four way intersection, a three way intersection, and a passing lane. Not all of these road features are present on each track layout; within the Track Layout Configurations section there will be a list of what testing opportunities are available in each layout.

There are multiple interactive elements of the track that provide opportunities to test the vehicles as well. This includes stop lights, stop signs, yield signs, and a railroad crossing. The elements have specific locations that they should be placed at in order for the virtual world to match the physical world properly.

This product utilizes a free software package that creates a virtual world that is synchronized with the physical track. This virtual world displays the current track layout, interactive element locations, and the vehicle on the track.

# Physical Track

## Given Equipment

* Track Mats
* Barrier Fabric Pieces
* Single Pole Mounting Points
* Double Pole Mounting Points
* Interactive Elements
  + Stop Lights
  + Stop Signs
  + Yield Signs
  + 1 Railroad Crossing
* 1 Raspberry PI 4

## Track Layout Configurations

There are 5 track layouts with full operability outlined in this document. These layouts come with different testing opportunities for the vehicles. They also vary in sizes between 12x38ft and 12x16ft.

All tracks are divided into individual columns (1-19) each with a fixed number of rows. Between the different setups the columns will change, so it is important to be aware of which columns are being used in each setup. Each setup has instructions for which numbered columns are to be used out of each section: entrance, straightaway, four-way, roundabout, semicircle.

### Medium Four Way

A black race track with yellow lines

Description automatically generated

Track columns used:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Track Sections | Entrance | Straightaway | Four Way | Roundabout | Semicircle |
| Track Columns | 1-5 | 6-7 | 10-16 | X | 17-19 |

Functionality:

* + Four-way intersection
    - With stop lights
    - With stop signs
  + Three-way intersection
    - With stop lights
    - With stop signs
  + Passing lane
  + Railroad crossing

### Small Four Way

A black road with yellow and blue lines

Description automatically generated

Track columns used:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Track Sections | Entrance | Straightaway | Four Way | Roundabout | Semicircle |
| Track Columns | 1-5 | X | 10-16 | X | 17-19 |

Functionality:

* + Four-way intersection
    - With stop lights
    - With stop signs
  + Three-way intersection
    - With stop lights
    - With stop signs
  + Passing lane
  + Railroad crossing

### Medium Roundabout

A black race track with a circle

Description automatically generated

Track columns used:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Track Sections | Entrance | Straightaway | Four Way | Roundabout | Semicircle |
| Track Columns | 1-5 | 6-7 | X | 10-16 | 17-19 |

Functionality:

* + Roundabout
    - With yield signs
  + Three-way intersection
    - With stop lights
    - With stop signs
  + Passing lane
  + Railroad crossing

### Small Roundabout

A black and yellow road with a circle

Description automatically generated with medium confidence

Track columns used:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Track Sections | Entrance | Straightaway | Four Way | Roundabout | Semicircle |
| Track Columns | 1-5 | X | X | 10-16 | 17-19 |

Functionality:

* + Roundabout
    - With yield signs
  + Three-way intersection
    - With stop lights
    - With stop signs
  + Passing lane
  + Railroad crossing

### Full Oval

A black race track with colorful lines

Description automatically generated

Track columns used:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Track Sections | Entrance | Straightaway | Four Way | Roundabout | Semicircle |
| Track Columns | 1-5 | 6-9 | X | X | 17-19 |

Functionality:

* + Three-way intersection
    - With stop lights
    - With stop signs
  + Passing lane
  + Railroad crossing

### Medium Oval

A black race track with colorful lines

Description automatically generated

Track columns used:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Track Sections | Entrance | Straightaway | Four Way | Roundabout | Semicircle |
| Track Columns | 1-5 | 6-7 | X | X | 17-19 |

Functionality:

* + Three-way intersection
    - With stop lights
    - With stop signs
  + Passing lane
  + Railroad crossing

### Small Oval

A black toy track with colorful lines

Description automatically generated

Track columns used:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Track Sections | Entrance | Straightaway | Four Way | Roundabout | Semicircle |
| Track Columns | 1-5 | X | X | X | 17-19 |

Functionality:

* + Three-way intersection
    - With stop lights
    - With stop signs

## Barrier Configuration

Each barrier fabric piece is ~60 in long with a few smaller options for filling in smaller gaps. These fabric pieces are slid onto wooden dowels of a length of about 9 in. Then, the dowels are attached to mounting points that are either for one or two poles. The single and double mounting points can be seen below in Figure 1.

A black object on a white surface

Description automatically generated

Figure 1: Single and Double Mounting Points

Dowels attached to a single mounting point do not have attached fabric pieces; dowels attached to a double mounting point have attached fabric pieces. On average, there are about 5 single mounting points between two double mounting points, though 4 should be assumed when setting up so as to save time if the fabric cannot reach. A complete setup barrier is shown in Figure 2.

A white fence on a grey surface

Description automatically generated with medium confidence

Figure 2: Barrier Example

Put enough barriers around the edges of the track to ensure the car cannot escape.

## Interactive Element Configuration

### Stop Sign

The stop sign interactive elements are assembled using the one tenth scale stop signs as well as one single mounting point. The post attached to the stop sign fits snugly into the single mounting point and can be rotated to the desired angle. The stop signs can be implemented at all mounting points in the entrance, roundabout, and four-way section.

### Yield Sign

Similar to the stop sign, the yield sign interactive elements are assembled using the one tenth scale yield sign as well as a single mounting point. The post attached to the yield sign fits snugly into the single mounting point and can be rotated to the desired angle. The yield signs can be implemented at all mounting points in the entrance, roundabout, and four-way section.

### Stop Lights

The stop light interactive elements are assembled with an ESP32 receiver, LEDs, as well as one single mounting point. The post attached to the stop light fits snugly into the single mounting point and can be rotated to the desired angle. The stop lights can be implemented at all mounting points in the entrance and four-way section.

### Railroad Crossing

The railroad crossing interactive element works with a servo motor and an ESP32 receiver. Each side of the crossing is attached to one single mounting point. The crossing can only be implemented on the straightaway section of the track.

# Software Implementation

This track requires multiple software products to run as intended.

## Software Products

In order to run the track, the following software products in Table 1 are required. This table also includes links to download the products.

Table 1: Required Software to Run Track

|  |  |
| --- | --- |
| Software Product | Website to download |
| ROS Humble Desktop Package | <https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html> |
| NoMachine | <https://www.nomachine.com/> |
| VS Code | <https://code.visualstudio.com/> |
| Linux 22.04 | <https://releases.ubuntu.com/jammy/> |
| Docker | <https://docs.docker.com/get-docker/> |
| Core Files | <https://github.com/UVMAirLabTrack/Track23> |
|  |  |

## Launch Files

All contained under the world\_gen package.

all\_viewer\_launch: This boots up the visualizer node, with all marker types enabled. Including the stoplights, signs, train crossing, and dragon. Also boots up the odom\_to\_map\_key executable to transform the car odometry to the map. And the Gen\_v2 to create the world mesh

Within, the various labelled publishers will display status messages.

Packages utilized:

• World\_gen: Four\_Way\_marker

• World\_gen: Three\_Way\_marker

• World\_gen: Stop\_sign

• World\_gen: Yield\_Sign

• World\_gen: Train\_marker

• Map\_transforms: Odom\_to\_map\_key

• World\_gen: Gen\_v2

• Xcore2: pub\_All\_pose

Track\_launch: Loads up the three\_way, four\_way and train control signal publishers, As well as the ESP\_serial node, which performs serial to wireless communications. This file is intended to be run on the Pi, or whatever computer you are utilizing to connect to the ESP network. Published state messages are shown in the terminal window.

Packages utilized:

• Four\_way\_light: gen

• three\_way\_light: gen

• train\_crossing: gen

• direct\_io: esp\_serial

Train\_viewer\_launch: Similar to All\_viewer\_launch, except only the train crossing markers are loaded.

Light\_viewer\_launch: similar to all\_viewer\_launch, except only the 3 and 4 way lights are loaded.

At present, all viewer launch files utilize the same RVIZ configuration file, in the future it is suggested to build thinner and lighter versions for each configuration.

## Interactive Controls

### Four\_way\_light

Code File: pub.py

Ros Build name: pub

Published topic(s): four\_way\_state

Subscription topic(s): None

Reference files: light\_states\_4.txt

This package contains a single core file, “pub”

The function is to parse the reference file of light states, and pass them along the topic in the structure of a 4 integer array to control the stoplight in the real and virtual space.

That array looks like: [Light1, Light2, Light3, Light4]

At present only 4 states are utilized, with the rest existing for expansion.

They are:

• 0: All Lights Off

• 1: “Red” Uses a pattern of the Red, White, and Blue LED’s illuminated

• 2: “Yellow” Only the yellow LED is illuminated

• 3: “Green” The Green and Blue LED’s are illuminated

To change the light patterns: enter the CoreRaspberry/control folder, and edit “light\_tates\_4.txt” Opening the file will look something like the figure below:

The system takes the first 4 numbers in a row, and sets the control variables for the lights, publishes, then waits the number of seconds specified by the 5th number in the row. Then moves to the next line. Then cycles back to the first entry when the last is reached.

### Three\_way\_light

Code File: pub.py

Ros Build name: pub

Published topic(s): three\_way\_state

Subscription topic(s): None

Reference files: light\_states\_3.txt

The function is to parse the reference file of light states, and pass them along the topic in the structure of a 4 integer array to control the stoplight in the real and virtual space.

That array looks like: [Light1, Light2, Light3, Light4]

At present only 4 states are utilized, with the rest existing for expansion.

They are:

* 0: All Lights Off
* 1: “Red” Uses a pattern of the Red, White, and Blue LED’s illuminated
* 2: “Yellow” Only the yellow LED is illuminated
* 3: “Green” The Green and Blue LED’s are illuminated

To change the light patterns: enter the CoreRaspberry/control folder, and edit “light\_tates\_3.txt” Opening the file will look something like the figure below:

A computer screen shot of a number

Description automatically generated

The system takes the first 4 numbers in a row, and sets the control variables for the lights, publishes, then waits the number of seconds specified by the 5th number in the row. Then moves to the next line. Then cycles back to the first entry when the last is reached.

### Train\_crossing

Code File: pub.py

Ros Build name: pub

Published topic(s): train\_crossing\_state

Subscription topic(s): None

Reference files: train\_states.txt

The function is to parse the reference file of crossing states, and pass them along the topic in the structure of a 4 integer array to control the barrier in the real and virtual space.

That array looks like: [Barrier1, Barrier2, Barrier3, Barrier4]

Each barrier state refers to an angle in degrees. With 0 being down, angles up to 180 are possible, but will likely run into obstruction by the fabric “Lidar fence”.

To change the barrier patterns: enter the CoreRaspberry/control folder, and edit “train\_states.txt” Opening the file will look something like the figure below:

A screen shot of a computer screen

Description automatically generated

The system takes the first 4 numbers in a row, and sets the control variables for the barriers, publishes, then waits the number of seconds specified by the 5th number in the row. Then moves to the next line. Then cycles back to the first entry when the last is reached.

### Direct\_io

Code File: esp\_serial.py

Ros Build name: esp\_serial

Published topic(s): None

Subscription topic(s): four\_way\_state, three\_way\_state,train\_crossing\_state,aux\_state

Reference files: None

This is the bridge between ROS and the ESP interactive network. The code intakes the states above and concatenates the states into a single 16 index long integer array in the following order:

[four\_way\_state (1-4) ,three\_way\_state (1-4), train\_crossing \_state (1-4),aux\_state(1-4)]

Then this array is broadcast via USB to all available ttyUSB devices listed at the time of activation, with utf-8 encoding.

This code can either be used for direct serial communication to endpoint nodes. Or via the ESP32 sender for wireless communications (highly recommended)

#### Additional code files in package:

There are also code files for and from prior testing, they are included as bonus items, but not intended for core function, hence will only be lightly documented and function is not guaranteed.

Those files are:

* direct\_master: interactive terminal designed to manually control lights, train ect ect. Prompts user.
* direct\_light: A pared down version of direct master, for lights only
* direct\_servo: a pared down of direct master for train crossing only
* testpub: broadcasts a series of test values to the four states received by esp\_serial.

## Virtual Space

### World Gen

This is the big package. The entire visualizer system for the track is located within, with exception for some common functions, and the map\_transforms utilized for odometry.

Code File: four\_way\_marker.py

Ros Build name: four\_way\_marker

Published topic(s): fourway\_(1,2,3,4)

Subscription topic(s): four\_way\_state, custom\_poses, marker\_loc

Reference files: light.stl, light\_pose.txt

This code file functions as one of many bridges between the track control states, and RVIZ.

At it’s core, there are two core behaviors.

The first, is the combined substriptions to custom\_poses and marker\_loc. (see these packages for their core function)

The system also parses the assigned \_pose.txt file to make corrections to the mesh. In practice this file should be left with 0 pose X,y,z and 0 quaternion angles. This is merely to adjust meshes who have poorly aligned coordinate systems.

Upon receiving the location poses via custom\_poses, the system then parses the marker\_loc subscription to look for any locations bearing the “fourway\_1” through 4 tag, then searches for the correct zone, and location for that tag. And assigns the marker pose to the coordinates contained within.

The second utilizes the subscription to “four\_way\_state” . When receiving the integer control states, the system performs a conversion from state value, to RGB color via an internal table. Then publish the marker mesh along “fourway\_1” through 4 to RVIZ for visualization.

Code File: three\_way\_marker.py

Ros Build name: three\_way\_marker

Published topic(s): threeway\_(1,2,3,4)

Subscription topic(s): three\_way\_state, custom\_poses, marker\_loc

Reference files: light.stl, light\_pose.txt

This code file functions as one of many bridges between the track control states, and RVIZ.

At it’s core, there are two core behaviors.

The first, is the combined subscriptions to custom\_poses and marker\_loc. (see these packages for their core function)

The system also parses the assigned \_pose.txt file to make corrections to the mesh. In practice this file should be left with 0 pose X,y,z and 0 quaternion angles. This is merely to adjust meshes who have poorly aligned coordinate systems.

Upon receiving the location poses via custom\_poses, the system then parses the marker\_loc subscription to look for any locations bearing the “threeway\_1” through 4 tag, then searches for the correct zone, and location for that tag. And assigns the marker pose to the coordinates contained within.

The second utilizes the subscription to “three\_way\_state” . When receiving the integer control states, the system performs a conversion from state value, to RGB color via an internal table. Then publish the marker mesh along “train\_1” through 4 to RVIZ for visualization.

Code File: train\_marker.py

Ros Build name: train\_marker

Published topic(s): train\_(1,2,3,4)

Subscription topic(s): train\_crossing\_state, custom\_poses, marker\_loc

Reference files: train\_barrier.dae,train\_barrier\_pose.txt

This code file functions as one of many bridges between the track control states, and RVIZ.

At it’s core, there are two core behaviors.

The first, is the combined subscriptions to custom\_poses and marker\_loc. (see these packages for their core function)

Upon receiving the location poses via custom\_poses, the system then parses the marker\_loc subscription to look for any locations bearing the “threeway\_1” through 4 tag, then searches for the correct zone, and location for that tag. And assigns the marker pose to the coordinates contained within.

The system also parses the assigned \_pose.txt file to make corrections to the mesh. In practice this file should be left with 0 pose X,y,z and 0 quaternion angles. This is merely to adjust meshes who have poorly aligned coordinate systems.

The final piece utilizes the trainc\_crossing\_state subscription. The states within correspond to a Y axis angle rotation, which represents the barrier’s present angle. To work with this, the system adds this angle, to the 3 Euler angles contained within the custom\_poses data. Then transforms to a quaternion for ROS publishing, over the train\_1 through 4 topics.

Code File: stop\_sign.py

Ros Build name: stop\_sign

Published topic(s): stop\_(1,2,3,4)

Subscription topic(s): custom\_poses, marker\_loc

Reference files: stop.dae,stop\_pose.txt

This code file functions as one of many bridges between the track control states, and RVIZ.

At it’s core, there are two core behaviors.

The first, is the combined subscriptions to custom\_poses and marker\_loc. (see these packages for their core function)

Upon receiving the location poses via custom\_poses, the system then parses the marker\_loc subscription to look for any locations bearing the “stop\_1” through 4 tag, then searches for the correct zone, and location for that tag. And assigns the marker pose to the coordinates contained within.

The system also parses the assigned \_pose.txt file to make corrections to the mesh. In practice this file should be left with 0 pose X,y,z and 0 quaternion angles. This is merely to adjust meshes who have poorly aligned coordinate systems.

After determining the proper coordinates, the system then publishes a marker over the stop\_1 through 4 topics to RVIZ.

Code File: yield\_sign.py

Ros Build name: yield\_sign

Published topic(s): yield\_(1,2,3,4)

Subscription topic(s): custom\_poses, marker\_loc

Reference files: yield.dae,yield\_pose.txt

See stop\_sign above, function is identical

Code File: dragon.py

Ros Build name: dragon

Published topic(s): dragon\_(1,2,3,4)

Subscription topic(s): custom\_poses, marker\_loc

Reference files: dragon.dae,dragon\_pose.txt

See stop\_sign above, function is identical

### Map\_transforms

Code File: odom\_to\_map\_key.py

Ros Build name: odom\_to\_map\_key

Published topic(s): startbox\_1, startbox\_2,car\_mesh\_1, car\_mesh\_2

Subscription topic(s): odom, reset\_car,custom\_poses,marker\_loc

Reference files: None

This file is one of the key bridges between real and virtual space.

At it’s core, there are two functions. One is to parse the custom poses and marker locations to determine the starting location of the vehicle. (Note, when first loaded, the vehicle will load to whatever it’s internal Pose is, Reset must be activated to move this). With this determined, a startbox mesh is published to that location.

The second is the node listens to the odometry data from the car with a 0.5 second frame buffer, strips out the pose data. Transforms this data from the base\_link reference used, to the map reference needed. Then stores this pose data.

When Reset\_Car is activated, the system captures the car’s current pose at the activation time, and saves this. Then subtracts this from the cars odom position. Adds the startbox location, then publishes this to the world as a simple mesh along car\_mesh\_1.

## Other

### Custom\_Messages

Code File: MarkerLoc.msg

Ros Build name: marker\_loc

Published topic(s): N/A

Subscription topic(s): N/A

Reference files:N/A

Contains a simple code file used to create a new topic message structure used by other packages. See Image below for structure.

A screen shot of a computer

Description automatically generated

Code File: WorldMarkers.msg

Ros Build name: custom\_poses

Published topic(s): N/A

Subscription topic(s): N/A

Reference files:N/A

Contains a simple code file used to create a new topic message structure used by other packages. See Image below for structure.

A screen shot of a computer

Description automatically generated

### X\_core2

Note: there are a handful of other files inside the xcore2 package, such as pose\_strip and formulas. These files are not directly run by any nodes, but contain techniques and formulas used by nearly all other files.

Code File: parse\_and\_pass.py

Ros Build name: pub\_All\_pose

Published topic(s): marker\_loc,custom\_poses

Subscription topic(s): none

Reference files: [world]\_markers.txt, [world]\_marker\_loc.txt

This is the core of the virtual system. As the name implies, this code file parses the text files, and passes the data along topics to other nodes.

The first file contains the respective locations of all possible marker locations in the selected track. In the structure

Note that currently the system is capable of passing quaternions, but all current subscription nodes utilize Euler angles in Radians.

Shown below is an example of that file, should any other locations be desired, simply add more coordinate points, with a zone and subzone. At this present, the Extra is currently unused.

A screenshot of a computer

Description automatically generated

The second file is used to assign the desired markers to the zones. In the structure:

Where the node name, due to how the system is structured, is also the marker’s published topic. The number at the end indicates which of the markers is being assigned to.

One example of this file is located below. To change the location a marker is assigned to, simply edit this file to change which marker is assigned to which location. Also, multiple markers can receive the same location. And simply disable them in RVIZ. Note that the names, zones, and subzones need to correspond EXACTLY to how they are listed in the file above, and the publisher topic names. Otherwise, the marker will appear at the origin.

Should this be a concern, run one of the individual marker publishers in a separate terminal, there are a series of print statements that will tell you what marker is going where, and if a zone or location can be found.

A screenshot of a computer program

Description automatically generated

### Marker mappings (not a code file)

A black background with red and green lines

Description automatically generated

A black circle with a circle in the middle

Description automatically generated with medium confidence

A black toy track with colorful lines

Description automatically generated with medium confidence

A black road with colorful lines

Description automatically generated

# Example Setup

## Example Physical Track Layout

For the example, the Medium Oval Track will be configured as it gives testing opportunities with four way intersections, as well as being the size of \_\_\_\_\_, which is not the largest the track can be. This allows for this example setup to be utilized by more places with varying size restraints.

## Example Hardware Setup

1. Acquire the foam mats required for the entrance section (1-5) the medium straight aways (6-7), and the semicircle (17-19).

2. Assemble the Medium Oval Track layout

3. Erect single mounting points and dowels by attaching the black mounting point magnet to the washers and inserting the dowels into the points if not done so already. For every 4 single mounting points, a space should be left to allow for the double mounting points.

4. Place all double pointing points in the spaces left between every 4 single mounting points.

5. Insert the dowels attached to the barrier fabric into the double mounting points and spin the dowel to ensure that the barrier fabric is taught.

6. Place stop signs upon the four washer mounting points that exist within the entrance section.

7. Place the railroad crossing barriers upon the four washer mounting points that exist within the straight away section.

8. Attach a 10ft ethernet cable to the railroad crossing barrier and run the cord under the nearest seam of the foam mats. Bring the cord to the middle of the track where there are no foam mats and plug it into the raspberry pi. Repeat for every railroad crossing barrier, using a separate 10ft ethernet cable for each element.

## Example Software Setup

To launch the example track. Open NoMachine from a computer connected to the same network as the Raspberry PI, and connect to “Track Pi” remotely.

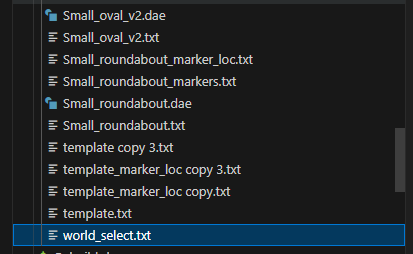
Once inside, to launch the track nodes. Open a new terminal window. And run the command

Ros2 launch world\_gen track\_launch.py

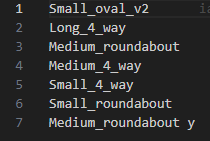
This command will begin broadcasting commands to control the nodes on the track.

To launch the virtual space, (NOT ON THE PI!) load a linux environment with the software package for the track.

Navigate to RaspberryCore/CoreRaspberry/worlds. And edit the world\_select.txt file.



Once inside, place the “y” next to the “medium\_oval” entry and save



Next: open a terminal window and run the following command:

Ros2 launch world\_gen all\_viewer\_launch.py

This will load RVIZ and the proper subscribers to view the world.

To add the car, and it’s odometry reset controller, run the following two commands in individual terminal windows.

Ros2 run map\_transforms odom\_to\_map

Ros2 run map\_transforms reset\_car

A new Red mesh should appear in RVIZ now, to reset the car’s odometry to it’s “start box” open the reset\_car terminal, and hit “Enter”

If for some reason communications are not established, it’s likely that the ros domain id is not correctly set. To fix this, add the following below, in front of the commands listed above

ROS\_DOMAIN\_ID=32 [command here]

# Future Work

## More Track Layouts

There is potential for two more tracks to be added with the following track columns:

Large Four Way

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Track Sections | Entrance | Straightaway | Four Way | Roundabout | Semicircle |
| Track Columns | 1-5 | 6-7 | 10-16 | X | 17-19 |

Large Roundabout

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Track Sections | Entrance | Straightaway | Four Way | Roundabout | Semicircle |
| Track Columns | 1-5 | 6-7 | X | 10-16 | 17-19 |