6.141/16.405 Robotics: Science and Systems

Final Challenge 2020

You've localized the car, made use of the camera, planned your own paths, and driven the car autonomously. Now it's time to put your knowledge to the test; it's time for the final challenge!

This year, the final challenge will be done in a photo-realistic simulated environment. You will be controlling a simulated car with applied car dynamics, and accessible lidar, imu, and camera data.

The final challenge is divided into 2 parts:

1) Final Race
2) Fast Obstacle Avoidance

The **Final Race** has the following objectives:

1) Program the car to follow a predefined trajectory using particle filter localization and pure pursuit control. We will provide a race map of the simulated environment.
2) Go as fast as you can!
3) Avoid collisions

Fast Obstacle Avoidance has the same objectives and uses the same map as the Final Race, but will include *randomly generated* static obstacles that the car will also need to detect and avoid in real-time.

Deliverables

Assignment	Due Date
Mock Race (optional)	**Monday, May 4th @ TBD**
RACE DAY	**Wednesday, May 6th @ 1-5pm** ● Final Race @ 1-3pm ● Fast Obstacle Avoidance @ 3-5pm
Final Presentation or Final Report (pick one and make sure to include videos from RACE DAY)	**Monday, May 11th @ 1-3pm** ● Can be Pre-Recorded or Live

Slides

The Final Challenge will be announced **Wednesday, April 22nd @ 1pm**. View the slides here.

Final Challenge Overview

From the previous labs, you should already have a working pure pursuit controller that is able to follow a path. However, there is still a lot to be tuned for a successful race, especially with the new car dynamics. You will need to test various paths, as well as make modifications to your controller, so be sure to begin testing early.

As before, you may use the instructor solution for particle filter localization from Lab 5. When moving quickly, you want to be sure your localization is running at as close to real-time as possible.

Challenge Details

The goal of the race is to tune your pure pursuit controller to work at faster speeds.

You will be required to complete a loop around the provided map as quickly as possible.
The map below (Figure 1) shows the environment you will be racing in.

Marked in red is one of the courses you may choose to follow as fast as possible. Your time will be determined based on measurements of your car crossing the blue lines. ~~You can start wherever you want.~~ Your car will be automatically spawned a short distance before the start line. You are encouraged to accelerate to your target speed before crossing the blue line marking the start.

Figure 1: Race Map and Track

Logistics

On race day, your team will have the chance to race 3 laps for each challenge (6 laps total) - the best lap from each challenge will be used together to determine your final score.

Each car will race individually! You do not need to worry about avoiding other cars. That being said, you are required to use a safety controller during all parts of the race. Drive this car as carefully as you would the actual RACECAR. Points will be deducted for collisions!

Final Race

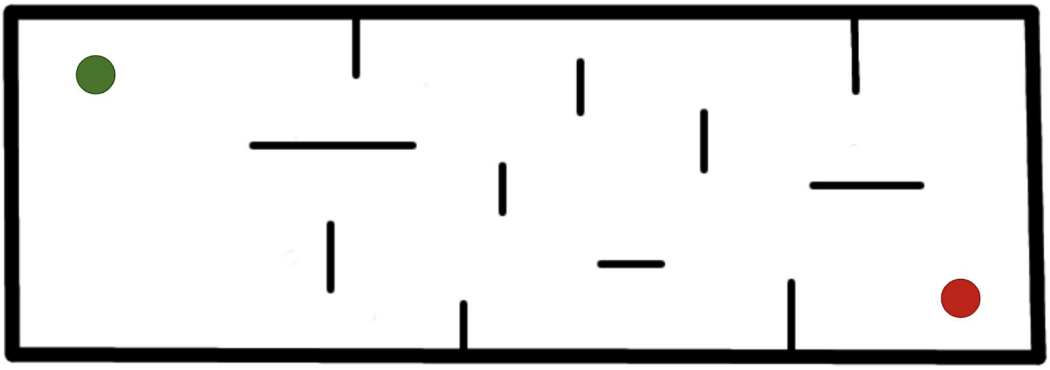
There will be no obstacles on the road. Your goal is to complete the track as fast as possible with little to no collisions. You may tune your predefined trajectories and controllers as much as needed in order to complete the track. The race track will NOT change on race day, so tune your parameters to your hearts' content!

Fast Obstacle Avoidance

The simulator will randomly spawn a number of static obstacles on the road. ~~The race track and map will be the same as the one used in the Final Race.~~ The map will be the same as the one used in the Final Race, but the race track will be different. Your car will need to detect and avoid these obstacles in real-time. The goal is to complete the track as fast as possible with little to no collisions, but this time *while avoiding obstacles*.

The obstacles will have the following properties:

- All obstacles will be stationary
- No dead ends
- No obstacles will be within 1m of start or goal
- No obstacle will obscure more than 50% of the road's width



Example Road with Obstacles
(**Green**: Start, **Red**: Stop)

Final Race Scoring

Your score is proportional to your best lap time. For the Final Race, your score can be calculated in **one of two ways**, depending on which course you decide on using.

Scores below a `70` will result in a failure. Scores above a `100` are absolutely allowed!

Short Course

The grading formula is based on a lap time of `45 seconds` being a score of `100` and a time of `75 seconds` being a score of `70`. The explicit formula is shown below (equation 1).

$$Score = -1 * time\,(sec) + 145 \quad (1)$$

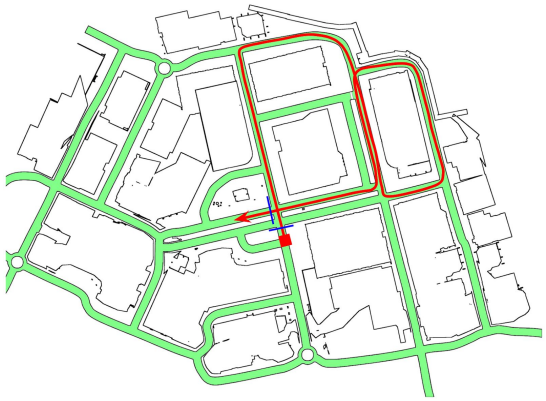


Figure 1. Short Race Map and Track

Long Course

The grading formula is based on a lap time of `55 seconds` being a score of `100` and a time of `105 seconds` being a score of `70`. The explicit formula is shown below (equation 2).

$$Score = -(3/5) * time\,(sec) + 133 \quad (2)$$

Figure 2. Long Race Map and Track

Point Deductions

You will lose points for the following events:

- `-10pts` for each collision

You will also earn an extra `+10 pts` for being the fastest team on race day.

Fast Obstacle Scoring

TBA

Getting Started

We will be using the TESSE simulator, developed in Unity3D by MIT Lincoln Laboratory and SPARK Lab. This simulator will enable you to visualize and drive the car within a photo-realistic simulation environment! Here is the link to the open-source code, if you are interested: https://github.com/MIT-TESSE/. We will be using a modified version of this simulator for RSS.

Unity Executable

Download the 2020 Final Challenge Unity executable **corresponding to your host OS** here. Go to Latest, and get the latest one (highest version number). Unzip it.

If you are using a Virtual Machine, do not try to run Unity on your VM. It will most likely be too slow. Instead, download and run the executable on your local machine. It will communicate with your VM through a local network.

ROS Bridge

To communicate with Unity, you will need a ROS bridge. The ROS bridge will play the role of the `racecar_simulator` package, to provide you with the standard rostopics that you are already familiar with from previous labs.

Download and install the ROS bridge by following the instructions here. Make sure to follow the instructions carefully, you are installing a python package as a submodule in the `python/` directory.

You should not need to modify the source code to get it working, but feel free to poke around and change parameters in the provided launch and parameter files. Everything ROS-related is in the `ROS/` subdirectory.

Once you have the ROS bridge cloned and built in your catkin workspace, you are ready to run the simulator! Verify that your downloaded copy of the simulator binary works by running it. You can do this by double-clicking on the binary/application/executable in the folder outside of your VM. Remember, the simulator is running on your host machine, not inside the racecar VM. The ROS bridge, which is inside the VM, uses the network to communicate with the simulator on your host.

NOTE: in Linux the executable might not be marked "executable" by the OS. In this case, open a terminal and run:

```
chmod +x tesse_exectuable_name.x86_64
```

Once you launch the executable on your host machine, you should be able to drive the car around in the simulated environment with your keyboard. At this point you won't be able to get data to/from the simulator via ROS just yet; we need to set up the network between your VM and the simulator.

Networking Setup

Only go through these steps if you are running linux on a VM. **You do not need to set up networking if you are running Linux natively.** If you are running native Linux, go directly to the "Testing your setup" section of this start guide.

Change your network adapter setting from "bridged" or "NAT" to "host-only". Unfortunately, you will not have access to the internet in "host-only" mode, so if you need internet access, switch back to NAT. Be sure to switch back to "host-only" in order to use the simulator.

After you have switched to **host-only mode**, find your VM and your host IP addresses:

- In your vm, run `hostname -I`. This is your IP address for your vm.
- In your host machine, follow the instructions for your specific host OS.
    - MacOS:
        - In a terminal, run `ifconfig | grep XXX.XXX.XXX`, where `XXX.XXX.XXX` is the first three entries vm ip. If your vm IP is `172.16.18.128`, you would run `ifconfig | grep 172.16.18`
        - In the line that is returned, the number with the matching subnet is your host machine IP.
            - For example: on my machine, my vm IP is `172.16.18.128` and my host IP is `172.16.18.1`.
    - Windows:
        - Press **(windows key) + R** and type "cmd.exe" into the textbox to open a command prompt (cmd.exe)
        - At the shell prompt, type `ipconfig`
        - The output of `ipconfig` will list various network interfaces and their associated IP addresses. Look out for an entry that looks something like "Ethernet Adapter ??? (?? VMware ??)" and make note of the IP address listed under "IPv4 Address". If it has the form `192.x.x.x` and the three first

numbers match your VM's IP address, then you can be sure that this is the IP address you want to use as your host machine's IP address.

Once you have both of your IP addresses, open the launch file at `~/catkin_ws/src/tesse-ros-bridge/ROS/launch/tesse_bridge.launch`. Here, you will need to make two changes.

- Set the default argument for `sim_ip` to your host IP
- Set the default argument for `self_ip` to your VM IP

Testing your setup

Troubleshooting tips

- If you get any import errors when running the launch file, change your network adapter to NAT, run the below command, then change your network adapter back to host-only

```
pip install empy opencv-python PyYAML scipy
```

- If you get the following error message after initialization:

```
('TESSE_ROS_NODE: clock_cb error: ',
AttributeError("'NoneType' object has no attribute
'metadata'",))
```

  This means that the simulator isn't responding to requests. Wait a few seconds; the simulator sometimes takes a long time to set up the sensors. At some point the simulator will unfreeze and you'll stop getting the error; everything should be good. If this continues ad infinitum and the simulator maintains a high (or crazy low) fps, contact us.

- If you're getting an error like above but with `image_cb` instead of `clock_cb`, try restarting your simulator. Sometimes it doesn't like it when you kill the ROS bridge and restart it soon after.
- If you're on Windows with a VM and you're not seeing data published to `/tesse/imu` or `/tesse/odom`, set `use_broadcast` to `true` in the launch file.
- If you're running native Linux with no VMs involved

To test if your networking is set up properly, launch the unity executable in your host machine.

- MacOS
  - Navigate to your unzipped unity executable:

```
cd tesse_rss_vXX/tesse_rss_vXX.app/Contents/MacOS
```

  - From there, run `./TESSE --client_ip_addr *VM IP*`
- Linux
  - Navigate to your unzipped unity executable

```
cd tesse_rss_vXX
```

  - From there `chmod +x tesse_rss_vXX.x86_64`

```
./tesse_rss_vXX.x86_64
```

- Windows
  - Open a command prompt by pressing (**windows key) + R** and typing `cmd.exe`.

    - In the command prompt, navigate to the TESSE executable's directory using the "cd" command
        - `cd C:\Users\YourUserName\...???...`
    - Start TESSE and provide your VM's IP address on the command line as follows:
        - `.\TESSE --client_ip_addr *VM_IP*`
    - Note that you may also use Powershell in Win10 if you want some Linux-esque commands

At this point, you should see the sim window with the car, and you can move this car around using the WASD keys.

On your VM

- Launch the ros node: `roslaunch tesse_ros_bridge tesse_bridge.launch`
- Launch rviz: `cd ~/racecar_ws/src/tesse-ros-bridge/ROS/`
  `rviz -d rviz/tesse_rss.rviz`
    - In rviz, make sure that you can see the laser scan from the simulator, and update that it moves around when you move your car around.
- Check metadata: `rostopic echo /tesse/imu`
    - If you get a clock error, check that you launched the executable with the `--client_ip_addr` flag and that you set it to the VM IP

Final Challenge Repository

Fork this `final_challenge/` repository to your team's github organization.
Make this repository public. The test server will need to be able to access this repo to test your code.

Please also add your team's `safety_controller/`, `localization/`, `path_planning/` and `visual_servoing/` packages as github submodules to your final challenge repository.

The test server will be running `git clone --recurse-submodules` on these packages.
Therefore, do not copy-and-paste these packages, or we will not be able to run your code properly.

The file structure of your `final_challenge/` repository should be organized as follows:

```
final_challenge/
  - final_challenge/
      - params/
          - ...
      - launch/
          - final_race.launch
          - fast_obstacle.launch
          - ...
      - src/
          - ...
      - maps/
```

```
            -  ...
        -  results/
            -  log.txt (more details coming soon)
    -  safety_controller/
    -  vision_modules/
    -  localization/
    -  path_planning/
```

Note that it is important that you **include the two launch files marked in bold text above**. These should bring up the entire system (so you don't have to manually launch anything except a single launch file pertaining to the challenge -- this includes safety controller, localization, path planning and following, and all other modules you need). Feel free to organize the contents of the directories marked with "..." however you want.

Topics

Much of the sensor data you would have been able to use on the Racecar are also provided by the TESSE simulator. This section details what ROS topics provide this sensor data.

Subscribable Topics

- `/tesse/imu`
- `/tesse/odom`
- `/tesse/hood_lidar/scan`
- `/tesse/trunk_lidar/scan`

- `/tesse/left_cam/image_raw`
- `/tesse/left_cam/camera_info`
- `/tesse/right_cam/image_raw`
- `/tesse/right_cam/camera_info`
- `/tesse/third_person/image_raw`
- `/tesse/third_person/camera_info`
- `/tesse/seg_cam/image_raw`
- `/tesse/seg_cam/camera_info`
- `/tesse/depth_cam/image_raw`
- `/tesse/depth_cam/camera_info`

- `/tf`
- `/tf_static`

Publishable Topics

- `/tesse/drive`

Important Topic Details

- `/tesse/imu`
    - angular_velocity

    - linear_acceleration
    - http://docs.ros.org/melodic/api/sensor_msgs/html/msg/Imu.html
    - **Note:** this is currently noiseless. A future build will make this noisy to prevent accurate dead-reckoning localization.
- `/tesse/.../scan`
    - angle_min
    - angle_max
    - angle_increment
    - range_min
    - range_max
    - ranges
    - http://docs.ros.org/melodic/api/sensor_msgs/html/msg/LaserScan.html
- `/tesse/odom`
    - child_frame_id
    - pose
    - twist
    - http://docs.ros.org/melodic/api/nav_msgs/html/msg/Odometry.html
    - **NOTE:** This is ground truth odometry! You won't have access to this on Race Day.
- `/tesse/drive`
    - velocity
    - steering_angle
    - http://docs.ros.org/jade/api/ackermann_msgs/html/msg/AckermannDriveStamped.html

Parameters

The parameters in `params/` will affect the sensors onboard the car. You'll see yaml files for each camera and LiDAR sensor; you may freely modify the parameters for each sensor as you see fit. **DO NOT** modify the LiDAR position and orientation parameters (except for fun). We will overwrite those changes on Race Day. You may change the LiDAR intrinsics (angles, number of beams, etc).

The launch file `tesse_bridge.launch` has many parameters you may change as well. Look through the arguments at the top of the file. Here are the important ones:

- `teleop`
    - If true, a new terminal will pop open that allows you to control the car by using WASD on your keyboard.
    - Unlike normal keyboard control with the executable, this publishes AckermannDriveStamped messages (just like your code!) which the interface then sends to the simulator.
    - This is mainly proof that you can control the car via ROS. **You do not need this for keyboard control**, and in fact should never need to turn it on. Stick to driving directly with the simulator.
- `sim_ip`
    - Set this to the IP of your host OS as viewed from the vm. More details in the "Networking Setup" section.
- `self_ip`

    - Set this to the IP of your vm as viewed from the host OS. More details in the "Networking Setup" section.
- `use_broadcast`
    - If true, the interface will listen to UDP high-rate packets on the broadcast IP. This really only works if you're running Linux natively (no vm).
- `publish_stereo_cams`
    - If true, the interface will publish left and right camera images to ROS. Disable unless you're doing stereo VIO.
- `publish_mono_stereo`
    - If true, the interface will publish left and right camera images to ROS as single-channel (mono) images. Disable unless you're doing stereo VIO.
- `publish_segmentation`
    - If true, the interface will publish the segmentation camera data to ROS. Disable unless you're using that information for something really cool.
- `publish_depth`
    - If true, the interface will publish the depth map to ROS as a single-channel image. Disable unless you're using that information for something really cool.
- `publish_third_pov`
    - If true, the interface will publish the third-person camera that is used when you open the simulator to ROS.
- `publish_metadata`
    - If true, the interface will publish the agent metadata as a string. For RSS you won't need this.
- `publish_segmentation`
    - If true, the interface will publish the segmentation camera data to ROS. Disable unless you're using that information for something really cool.
- `publish_hood_lidar`
    - If true, the interface will publish the hood-mounted lidar scans to ROS.
- `publish_trunk_lidar`
    - If true, the interface will publish the trunk-mounted lidar scans to ROS.
- `speedup_factor`
    - Your system might have a slow framerate. If you want to record rosbags that aren't super slow, set this to > 1.0. This will effectively speed up the timestamps published by the simulator.
- `frame_rate, imu_rate, scan_rate`
    - Leave them as is unless you have a super powerful machine. They just control the rate at which the interface queries the simulator.
- `enable_collision`
    - If false, the interface will tell the simulator to turn off all collisions in the world. You will noclip through walls if you hit them. Useful for initial testing if you're running into things too often.
- `initial_scene`
    - Set to the scene you want the simulator to load into when the ROS bridge starts.
        - 1 - Windridge City Scene: Official test environment for the final race and final challenge

- 2 - XXX (Under Construction): Large open space, ideal for controller and obstacle testing
- 3 - XXX (Under Construction): Crowded urban environment, ideal for localization testing

Running the Test on the Server (Supercloud)

Your team will be able to upload your code to be tested on a supercloud server, to ensure that all teams are scored fairly, regardless of personal machine type.

The server will only read code from the `test` branch of the `final_challenge/` repository. Once code is committed, it will immediately begin running tests. Once the tests are completed, the results will be automatically emailed to your team. (**Subject to change: we are still finishing up this setup. It is possible there may be some small changes to this workflow. We will provide updates on this.**)

To ensure that your code can be run on the server, please do the following before committing:

1. Make sure that your `final_challenge/` repo matches the file structure described above

When your code is committed to the `test` branch, the server will run the following commands:

1. `git pull`
2. `git clone --recurse-submodules …`
3. `roslaunch final_challenge final_race.launch`
4. `roslaunch final_challenge fast_obstacle.launch`

Your code will be officially scored on Final Race Day. Any tests done beforehand will not count towards the final score.

Notes

- For the **Final Race**,
  - the race track and initial spawn point of the car will remain the same
- For **Fast Obstacle Avoidance**,
  - the number of obstacles will be randomly generated
  - the obstacle spawn locations will vary
- The server will automatically set the `scene` and `num_obstacles` parameters in your `params.yaml` file:
  - `scene: 3` (windridge city scene)
  - `num_obstacles: <random#>`
- Do not edit tesse_ros_bridge. We will have a separate ros bridge on the server side. Any edits made locally will not be applied.
- Do not attempt to hack the server. Please.

Race Day Logistics

TBD

Notes and Resources

Tuning Paths and Pure Pursuit

There are a lot of variables at play here - you can spend weeks simply trying out different paths and modifications to your PP controller and evaluating how they interact with each other. You are free to explore different methods of improving both your chosen path and your controller, but here are a couple starting questions to think about:

- What path is the “best” for a time trial for the Final Race?
- At what point do you want to be driving quickly, and at what point driving slowly?
- How does look-ahead distance affect your performance? During turns? At speed?

Rostopic List

Run `rostopic list` to get an idea of what car data is published and available for you to use. Note that the `ground_truth` topic will NOT be available on the official test server, and is only available to you locally and in the resulting rosbag from the test server, so that you may use it for debugging purposes (and for post-mortem analysis). Please do not rely on ground truth data in your algorithm.

Realistic Physics

The simulated car uses a realistic physics engine. Therefore, you will need to account for delays in acceleration and deceleration forces when determining velocity changes. Furthermore, the tilting of the car may add noise to the lidar data received. Do not assume a perfect simulation environment.